Anne Tasso

Un best-seller qui a déjà conquis plus de 50 000 étudiants !

Le livre de JAVA premier langage

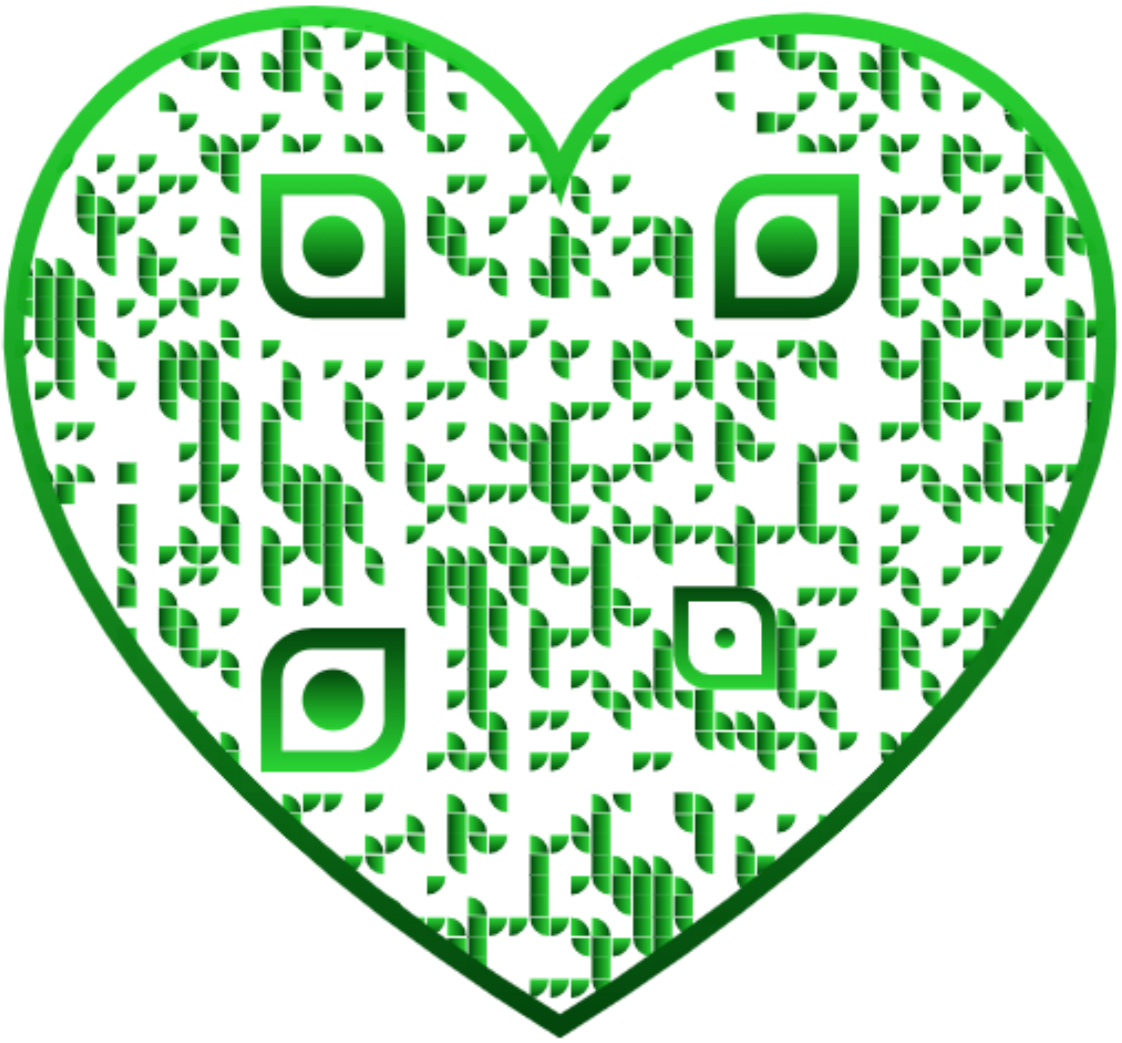
Avec 109 exercices corrigés

11e édition

Corrigé du projet et des exercices
Code source des exemples du livre

EYROLLES

Copyright © 2015 Eyrolles



Le livre de JAVA
premier langage

11e édition

Apprendre Java en douceur

Vous avez décidé de vous initier à la programmation et souhaitez opter pour un langage largement utilisé dans le monde professionnel ? Java se révèle un choix idéal comme vous le constaterez dans ce livre conçu pour les vrais débutants en programmation.

Vous apprendrez d'abord, à travers des exemples simples en Java, à maîtriser les notions communes à tous les langages : variables, types de données, boucles et instructions conditionnelles, etc. Vous franchirez un nouveau pas en découvrant par la pratique les concepts de la programmation orientée objet (classes, objets, héritage), puis le fonctionnement des librairies graphiques AWT et Swing (fenêtres, gestion de la souris, tracé de graphiques). Cet ouvrage vous expliquera aussi comment réaliser des applications Java dotées d'interfaces graphiques conviviales grâce au logiciel libre NetBeans. Enfin, vous vous initierez au développement d'applications avec l'interface Android Studio.

Chaque chapitre est accompagné de deux types de travaux pratiques : des exercices, dont le corrigé est fourni sur l'extension web du livre, et un projet développé au fil de l'ouvrage, qui vous montrera comment combiner toutes les techniques de programmation étudiées pour construire une véritable application Java.

À qui s'adresse ce livre ?

- Aux étudiants de 1er cycle universitaire (IUT, Deug...) ou d'écoles d'ingénieurs
- Aux vrais débutants en programmation : passionnés d'informatique et programmeurs autodidactes, concepteurs de sites Web souhaitant aller au-delà de HTML et JavaScript, etc.
- Aux enseignants et formateurs recherchant une méthode pédagogique et un support de cours pour enseigner Java à des débutants

Sur le site www.annetasso.fr/java

- Consultez les corrigés du projet et des exercices
- Téléchargez le code source de tous les exemples du livre
- Dialoguez avec l'auteur

Maître de conférences à l'université Paris-Est Marne-la-Vallée, **Anne Tasso** enseigne le langage Java en formation initiale et continue, au sein du département MMI (Métiers du Multimédia et de l'Internet) de l'IUT de Marne-la-Vallée. Son public universitaire est essentiellement constitué de débutants en programmation, ce qui lui a permis d'élaborer une méthode pédagogique structurée et imagée. Son objectif est d'expliquer, avec des mots simples, les techniques de programmation jusqu'à un niveau avancé.

Sommaire

Introduction. Qu'est-ce qu'un programme ? • Construire un algorithme • Premier programme en Java • Exécution du programme • **Outils et techniques de base.** Stocker une information • Données, variables et opérateurs • Entrées-sorties • Instructions et boucles • **Initiation à la programmation orientée objet.** De l'algorithme paramétré à l'écriture de fonctions • Classes et objets • Passage de paramètres par valeur et par référence • Héritage et polymorphisme • Interfaces • **Programmation orientée objet et interfaces graphiques.** Tableaux • Listes et dictionnaires • Archivage des données • Gestion des exceptions • Librairies AWT et Swing • Fenêtre, clavier et souris • Interface graphique avec NetBeans • Développer avec Android Studio.

www.editions-eyrolles.com

Copyright © 2015 Eyrolles

Le livre de JAVA premier langage

Avec 109 exercices corrigés

Copyright © 2016 Eyrolles.

CHEZ LE MÊME ÉDITEUR

Autres ouvrages sur Java

C. DELANNOY. – **Programmer en Java (9e édition)**.
N°14007, 2014, 940 pages.

C. DELANNOY. – **Exercices en Java (4e édition)**.
N°14009, 2014, 360 pages.

C. DELANNOY. – **Programmer en Java (6e édition)**. *Java 5 et 6*.
N°13443, 2012, 788 pages (format semi-poche).

J.-B. BOICHAT. – **Apprendre Java et C++ en parallèle (4e édition)**.
N°12403, 2008, 600 pages + CD-Rom.

A. PATRICIO. – **Java Persistence et Hibernate.**
N°12259, 2008, 364 pages.

E. PUYBARET. – **Bien programmer en Java 7.**
N°12794, 2012, 428 pages.

R. FLEURY. – **Les Cahiers du programmeur Java/XML.**
N°11316, 2004, 218 pages.

P. HAGGAR. – **Mieux programmer en Java**. *68 astuces pour optimiser son code*.
N°9171, 2000, 256 pages.

J.-P. RETAILLÉ. – **Refactoring des applications Java/J2EE**.
N°11577, 2005, 390 pages.

Autres ouvrages

C. DELANNOY. – **Programmer en Fortran**. *Fortran 90 et ses évolutions - Fortran 95, 2003 et 2008*.
N°14020, 2015, 454 pages.

C. DELANNOY. – **Le guide complet du langage C**.
N°14012, 2014, 844 pages.

C. DELANNOY. – **S'initier à la programmation et à l'orienté objet**. *Avec des exemples en C, C++, C#, Python, Java et PHP*.
N°14067, 2014, 382 pages.

G. DOWEK *et al.* – **Informatique et sciences du numérique**. *Manuel de spécialité ISN en terminale*.
N°13676, 2013, 354 pages.

G. DOWEK *et al.* – **Informatique pour tous en classes préparatoires aux grandes écoles**. *Manuel d'algorithmique et programmation structurée avec Python*.
N°13700, 2013, 408 pages.

Copyright © 2016 Eyrolles.

Anne Tasso

Le livre de JAVA premier langage

Avec 109 exercices corrigés

11e édition

Copyright © 2016 Eyrolles.

EYROLLES

ÉDITIONS EYROLLES
61, bd Saint-Germain
75240 Paris Cedex 05
www.editions-eyrolles.com

Copyright © 2016 Eyrolles.

En application de la loi du 11 mars 1957, il est interdit de reproduire intégralement ou partiellement le présent ouvrage, sur quelque support que ce soit, sans l'autorisation de l'Éditeur ou du Centre Français d'exploitation du droit de copie, 20, rue des Grands Augustins, 75006 Paris.
© Groupe Eyrolles, 2000-2016, ISBN : 978-2-212-14384-3

Avec cette onzième édition, je tiens à remercier tous mes nombreux lecteurs pour leurs félicitations qui me vont droit au cœur et leurs remarques toujours constructives.

Je remercie également tous mes étudiants, qui par leurs interrogations, leurs retours et leur curiosité m'ont permis d'écrire ce livre avec le souhait d'apporter des explications claires et précises.

Et enfin, un merci tout particulier à Antoine Derouin qui m'offre le temps de faire naître chaque livre avec beaucoup d'esprit, d'attention et de patience.

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

Copyright © 2016 Eyrolles.

Table des matières

Avant-propos – Organisation de l'ouvrage 1

Introduction – Naissance d'un programme 5

Construire un algorithme 5
Ne faire qu'une seule chose à la fois 6
Exemple : l'algorithme du café chaud 6
Vers une méthode 8
Passer de l'algorithme au programme 9
Qu'est-ce qu'un ordinateur ? 9
Un premier programme en Java, ou comment parler à un ordinateur 14
Exécuter un programme 22
Compiler, ou traduire en langage machine 22
Compiler un programme écrit en Java 22
Les environnements de développement 25
Le projet : Gestion d'un compte bancaire 26
Cahier des charges 26
Les objets manipulés 29
La liste des ordres 29
Résumé 31
Exercices 32
Apprendre à décomposer une tâche en sous-tâches distinctes 32
Observer et comprendre la structure d'un programme Java 32
Écrire un premier programme Java 33

Partie I
Outils et techniques de base

1 Stocker une information 37
La notion de variable 38
Les noms de variables 38
La notion de type 39
Les types de base en Java 40
Comment choisir un type de variable plutôt qu'un autre ? 44
Déclarer une variable 45

Copyright © 2016 Eyrolles.

L'instruction d'affectation ... 47
Rôle et mécanisme de l'affectation ... 47
Déclaration et affectation ... 48
Quelques confusions à éviter ... 50
Échanger les valeurs de deux variables ... 51
Les opérateurs arithmétiques ... 52
Exemple ... 52
La priorité des opérateurs entre eux ... 53
Le type d'une expression mathématique ... 54
La transformation de types ... 56
Calculer des statistiques sur des opérations bancaires ... 59
Cahier des charges ... 59
Le code source complet ... 62
Résultat de l'exécution ... 62
Résumé ... 63
Exercices ... 64
Repérer les instructions de déclaration, observer la syntaxe d'une instruction ... 64
Comprendre le mécanisme de l'affectation ... 64
Comprendre le mécanisme d'échange de valeurs ... 65
Calculer des expressions mixtes ... 66
Comprendre le mécanisme du cast ... 66
Le projet : Gestion d'un compte bancaire ... 67
Déterminer les variables nécessaires au programme ... 67

2 Communiquer une information ... 69
La bibliothèque System ... 69
L'affichage de données ... 70
Affichage de la valeur d'une variable ... 71
Affichage d'un commentaire ... 71
Affichage de plusieurs variables ... 71
Affichage de la valeur d'une expression arithmétique ... 72
Affichage d'un texte ... 73
La saisie de données ... 76
La classe Scanner ... 77
Résumé ... 81
Exercices ... 82
Comprendre les opérations de sortie ... 82
Comprendre les opérations d'entrée ... 82
Observer et comprendre la structure d'un programme Java ... 83
Le projet : Gestion d'un compte bancaire ... 84
Afficher le menu principal ainsi que ses options ... 84

Copyright © 2016 Eyrolles.

3 Faire des choix 85
L'algorithme du café chaud, sucré ou non 85
Définition des objets manipulés 86
Liste des opérations 86
Ordonner la liste des opérations 86
L'instruction if-else 89
Syntaxe d'if-else 89
Comment écrire une condition 90
Rechercher le plus grand de deux éléments 92
Deux erreurs à éviter 95
Des if-else imbriqués 96
L'instruction switch, ou comment faire des choix multiples 98
Construction du switch 98
Calculer le nombre de jours d'un mois donné 99
Comment choisir entre if-else et switch ? 102
Résumé 103
Exercices 104
Comprendre les niveaux d'imbrication 104
Construire une arborescence de choix 105
Manipuler les choix multiples, gérer les caractères 106
Le projet : Gestion d'un compte bancaire 107
Accéder à un menu suivant l'option choisie 107

4 Faire des répétitions 109
Combien de sucres dans votre café ? 110
La boucle do...while 111
Syntaxe 112
Principes de fonctionnement 112
Un distributeur automatique de café 112
La boucle while 119
Syntaxe 119
Principes de fonctionnement 119
Saisir un nombre entier au clavier 120
La boucle for 127
Syntaxe 127
Principes de fonctionnement 128
Rechercher le code Unicode d'un caractère de la table ASCII 128
Quelle boucle choisir ? 131
Choisir entre une boucle do...while et une boucle while 131
Choisir entre la boucle for et while 132
Résumé 132
Exercices 134
Comprendre la boucle do...while 134

Copyright © 2016 Eyrolles.

Apprendre à compter, accumuler et rechercher une valeur 135
Comprendre la boucle while, traduire une marche à suivre en programme Java 135
Comprendre la boucle for 136
Le projet : Gestion d'un compte bancaire 137
Rendre le menu interactif 137

Partie II
Initiation à la programmation orientée objet

5 De l'algorithme paramétré à l'écriture de fonctions 141
Algorithme paramétré 142
Faire un thé chaud, ou comment remplacer le café par du thé 142
Des fonctions Java prédéfinies 144
La bibliothèque Math 144
Exemples d'utilisation 146
Principes de fonctionnement 147
Construire ses propres fonctions 149
Appeler une fonction 150
Définir une fonction 151
Les fonctions au sein d'un programme Java 156
Comment placer plusieurs fonctions dans un programme 156
Les différentes formes d'une fonction 158
Résumé 161
Exercices 162
Apprendre à déterminer les paramètres d'un algorithme 162
Comprendre l'utilisation des fonctions 162
Détecter des erreurs de compilation concernant les paramètres ou le résultat d'une fonction 163
Écrire une fonction simple 164
Le projet : Gestion d'un compte bancaire 166
Définir une fonction 166
Appeler une fonction 166

6 Fonctions, notions avancées 167
La structure d'un programme 167
La visibilité des variables 169
Variable locale à une fonction 170
Variable de classe 173
Quelques précisions sur les variables de classe 175

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

Les fonctions communiquent ... 178
Le passage de paramètres par valeur ... 179
Le résultat d'une fonction ... 181
Lorsqu'il y a plusieurs résultats à retourner ... 183
Résumé ... 185
Exercices ... 186
Repérer les variables locales et les variables de classe ... 186
Communiquer des valeurs à l'appel d'une fonction ... 187
Transmettre un résultat à la fonction appelante ... 188
Le projet : Gestion d'un compte bancaire ... 188
Comprendre la visibilité des variables ... 188
Les limites du retour de résultat ... 189

7 Les classes et les objets ... 191
La classe String, une approche de la notion d'objet ... 191
Manipuler des mots en programmation ... 192
Les différentes méthodes de la classe String ... 194
Appliquer une méthode à un objet ... 203
Construire et utiliser ses propres classes ... 205
Définir une classe et un type ... 205
Définir un objet ... 209
Manipuler un objet ... 211
Une application qui utilise des objets Cercle ... 212
Résumé ... 216
Exercices ... 217
Utiliser les objets de la classe String ... 217
Créer une classe d'objets ... 218
Consulter les variables d'instance ... 218
Analyser les résultats d'une application objet ... 218
Le projet : Gestion d'un compte bancaire ... 221
Traiter les chaînes de caractères ... 221
Définir le type Compte ... 221
Construire l'application Projet ... 222
Définir le type LigneComptable ... 222
Modifier le type Compte ... 222
Modifier l'application Projet ... 223

8 Les principes du concept objet ... 225
La communication objet ... 226
Les données static ... 226
Le passage de paramètres par référence ... 229

Copyright © 2016 Eyrolles.

Les objets contrôlent leur fonctionnement ... 234
La notion d'encapsulation ... 235
La protection des données ... 235
Les méthodes d'accès aux données ... 237
Les constructeurs ... 243
L'héritage ... 246
La relation « est un » ... 246
Le constructeur d'une classe héritée ... 248
La protection des données héritées ... 250
Le polymorphisme ... 250
Les interfaces ... 252
Qu'est-ce qu'une interface ? ... 252
Calculs géométriques ... 254
Résumé ... 257
Exercices ... 258
La protection des données ... 258
L'héritage ... 260
Les interfaces ... 263
Le projet : Gestion d'un compte bancaire ... 264
Encapsuler les données d'un compte bancaire ... 264
Comprendre l'héritage ... 266

Partie III
Outils et techniques orientés objet

9 Collectionner un nombre fixe d'objets ... 271
Les tableaux à une dimension ... 272
Déclarer un tableau ... 272
Manipuler un tableau ... 274
Quelques techniques utiles ... 278
La ligne de commande ... 278
Trier un ensemble de données ... 283
Les tableaux à deux dimensions ... 291
Déclaration d'un tableau à deux dimensions ... 291
Accéder aux éléments d'un tableau ... 292
Résumé ... 299
Exercices ... 300
Les tableaux à une dimension ... 300
Les tableaux d'objets ... 301
Les tableaux à deux dimensions ... 301
Pour mieux comprendre le mécanisme des boucles imbriquées for–for ... 302

Copyright © 2016 Eyrolles.

Le projet : Gestion d'un compte bancaire ... 303
Traiter dix lignes comptables ... 303

10 Collectionner un nombre indéterminé d'objets ... 305
La programmation dynamique ... 305
Les listes ... 306
Les dictionnaires ... 311
Les streams et les expressions lambda ... 322
L'archivage de données ... 324
La notion de flux ... 324
Les fichiers textes ... 325
Les fichiers d'objets ... 329
Gérer les exceptions ... 334
Résumé ... 337
Exercices ... 339
Comprendre les listes ... 339
Comprendre les dictionnaires ... 341
Créer des fichiers textes ... 342
Créer des fichiers d'objets ... 344
Gérer les erreurs ... 344
Le projet : Gestion d'un compte bancaire ... 345
Les comptes sous forme de dictionnaire ... 345
La sauvegarde des comptes bancaires ... 346
La mise en place des dates dans les lignes comptables ... 346

11 Dessiner des objets ... 349
La bibliothèque AWT ... 349
Les fenêtres ... 350
Le dessin ... 352
Les éléments de communication graphique ... 358
Les événements ... 362
Les types d'événements ... 362
Exemple : associer un bouton à une action ... 363
Exemple : fermer une fenêtre ... 367
Quelques principes ... 368
De l'AWT à Swing ... 368
Un sapin en Swing ... 369
Modifier le modèle de présentation de l'interface ... 372
Résumé ... 379
Exercices ... 380
Comprendre les techniques d'affichage graphique ... 380
Apprendre à gérer les événements ... 381

Copyright © 2016 Eyrolles.

Le projet : Gestion d'un compte bancaire ... 385
Calcul de statistiques ... 385
L'interface graphique ... 386

12 Créer une interface graphique ... 389
Un outil d'aide à la création d'interfaces graphiques ... 389
Qu'est qu'un EDI ? ... 390
Une première application avec NetBeans ... 400
Gestion de bulletins de notes ... 410
Cahier des charges ... 411
Mise en place des éléments graphiques ... 413
Définir le comportement des objets graphiques ... 420
Un éditeur pour dessiner ... 433
Cahier des charges ... 434
Créer une feuille de dessins ... 435
Créer une boîte à outils ... 445
Créer un menu ... 451
Résumé ... 455
Exercices ... 455
S'initier à NetBeans ... 455
Le gestionnaire d'étudiants version 2 ... 457
L'éditeur graphique version 2 ... 461
Le projet : Gestion de comptes bancaires ... 463
Cahier des charges ... 463
Structure de l'application ... 465
Mise en place des éléments graphiques ... 467
Définition des comportements ... 470

13 Développer une application Android ... 475
Comment développer une application mobile ? ... 475
Bonjour le monde : votre première application mobile ... 476
L'application Liste de courses ... 490
Publier une application Android ... 508
Tester votre application sur un mobile Android ... 509
Déposer une application Android sur un serveur dédié ... 512
Résumé ... 523
Exercices ... 525
Comprendre la structure d'un projet Android ... 525
La liste des courses – Version 2 ... 527

Copyright © 2016 Eyrolles.

Annexe – Guide d'installations **531**
Extension Web 531
Le fichier corriges.pdf 531
L'archive Sources.zip 535
Le lien Java 535
Le lien NetBeans 535
Le lien Android Studio 535
Installation d'un environnement de développement 536
Installation de Java SE Development Kit sous Windows 536
Installation de Java SE Development Kit 8 sous Mac OS X 545
Installation de Java SE Develoment Kit 8 sous Linux 549
Installation de NetBeans sous Windows 2000, NT, XP, Vista et 7 550
Installation de NetBeans sous Mac OS X 10.7 et supérieur 555
Installation de NetBeans sous Linux 561
Utilisation des outils de développement 565
Installer la documentation en ligne 565
Développer en mode commande 565
Développer avec NetBeans 570
Développer des applications Android avec Android Studio 576

Index **589**

Copyright © 2016 Eyrolles.

Copyright © 2016 Eyrolles.

Avant-propos

Organisation de l'ouvrage

Ce livre est tout particulièrement destiné aux débutants qui souhaitent aborder l'apprentissage de la programmation en utilisant le langage Java comme premier langage.

Les concepts fondamentaux de la programmation y sont présentés de façon évolutive, grâce à un découpage de l'ouvrage en trois parties, chacune couvrant un aspect différent des outils et techniques de programmation.

Le chapitre introductif, « Naissance d'un programme », constitue le préalable nécessaire à la bonne compréhension des parties suivantes. Il introduit aux mécanismes de construction d'un algorithme, compte tenu du fonctionnement interne de l'ordinateur, et explique les notions de langage informatique, de compilation et d'exécution à travers un exemple de programme écrit en Java.

La première partie concerne l'étude des « Outils et techniques de base » :

- Le chapitre 1, « Stocker une information », aborde la notion de variables et de types. Il présente comment stocker une donnée en mémoire, calculer des expressions mathématiques ou échanger deux valeurs, et montre comment le type d'une variable peut influencer le résultat d'un calcul.
- Le chapitre 2, « Communiquer une information », explique comment transmettre des valeurs à l'ordinateur par l'intermédiaire du clavier et montre comment l'ordinateur fournit des résultats en affichant des messages à l'écran.
- Le chapitre 3, « Faire des choix », examine comment tester des valeurs et prendre des décisions en fonction du résultat. Il traite de la comparaison de valeurs ainsi que de l'arborescence de choix. Avec en exemple, la nouvelle structure de test `switch` de la version 7 de Java.
- Le chapitre 4, « Faire des répétitions », est consacré à l'étude des outils de répétition et d'itération. Il aborde les notions d'incrémentation et d'accumulation de valeurs (compter et faire la somme d'une collection de valeurs).

La deuxième partie, « Initiation à la programmation orientée objet », introduit les concepts fondamentaux indispensables à la programmation objet.

- Le chapitre 5, « De l'algorithme paramétré à l'écriture de fonctions », montre l'intérêt de l'emploi de fonctions dans la programmation. Il examine les différentes étapes de leur création.
- Le chapitre 6, « Fonctions, notions avancées », décrit très précisément comment manipuler les fonctions et leurs paramètres. Il définit les termes de variable locale et de classe, et explique le passage de paramètres par valeur.

Copyright © 2016 Eyrolles.

- Le chapitre 7, « Les classes et les objets », explique à partir de l'étude de la classe `String`, ce que sont les classes et les objets dans le langage Java. Il montre ensuite comment définir de nouvelles classes et construire des objets propres à l'application développée. Avec en exemple, une nouvelle façon de comparer des chaînes de caractères grâce à la nouvelle structure de test `switch` de la version 7 de Java.
- Le chapitre 8, « Les principes du concept d'objet », développe plus particulièrement comment les objets se communiquent l'information, en expliquant notamment le principe du passage de paramètres par référence. Il décrit ensuite les principes fondateurs de la notion d'objet, c'est-à-dire l'encapsulation des données (protection et contrôle des données, constructeur de classe) ainsi que l'héritage entre classes et la notion d'interfaces.

La troisième partie, « Outils et techniques orientés objet », donne tous les détails sur l'organisation, le traitement et l'exploitation intelligente des objets.

- Le chapitre 9, « Collectionner un nombre fixe d'objets », concerne l'organisation des données sous la forme d'un tableau de taille fixe.
- Le chapitre 10, « Collectionner un nombre indéterminé d'objets », présente les différents outils qui permettent d'organiser dynamiquement en mémoire les ensembles de données de même nature, notamment les nouvelles fonctionnalités de Java 8, à savoir les expressions lambda et les streams. Il est également consacré aux différentes techniques d'archivage et à la façon d'accéder aux informations stockées sous forme de fichiers.
- Le chapitre 11, « Dessiner des objets », couvre une grande partie des outils graphiques proposés par le langage Java (bibliothèques AWT et Swing). Il analyse le concept événement-action.
- Le chapitre 12, « Créer une interface graphique », expose dans un premier temps le fonctionnement de base de l'environnement de programmation NetBeans. Puis, à travers trois exemples très pratiques, il montre comment concevoir des applications munies d'interfaces graphiques conviviales.
- Le chapitre 13, « Développer une application Android », décrit comment créer votre toute première application Android, tout en expliquant la structure de base nécessaire au déploiement de cette application. Il présente ensuite le développement d'une application plus élaborée ainsi que sa mise à disposition sur un serveur dédié.

Ce livre contient également en annexe :

- un guide d'installation détaillé des outils nécessaires au développement des applications Java (Java, NetBeans), sous Linux, Mac OS X et sous Windows 2000, NT, XP et Vista ;
- toutes les explications nécessaires pour construire votre environnement de développement d'applications Java ou Android, que ce soit en mode commande ou en utilisant la plateforme ou Android Studio pour les applications sur smartphone ou tablette ;
- un index, qui vous aidera à retrouver une information sur le thème que vous recherchez (les mots-clés du langage, les exemples, les principes de fonctionnement, les classes et leurs méthodes, etc.).

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

Chaque chapitre se termine sur une série d'exercices offrant au lecteur la possibilité de mettre en pratique les notions qui viennent d'être étudiées. Un projet est également proposé au fil des chapitres afin de développer une application de gestion d'un compte bancaire. La mise en œuvre de cette application constitue un fil rouge qui permettra au lecteur de combiner toutes les techniques de programmation étudiées au fur et à mesure de l'ouvrage, afin de construire une véritable application Java.

Extension Web Les codes sources des exemples, des exercices et du projet sont téléchargeables depuis l'extension Web *www.annetasso.fr/Java* en cliquant sur le lien Sources.

Copyright © 2016 Eyrolles.

Copyright © 2016 Eyrolles.

Introduction

Naissance d'un programme

Aujourd'hui, l'informatique en général et l'ordinateur en particulier sont d'un usage courant. Grâce à Internet, l'informatique donne accès à une information mondiale. Elle donne aussi la possibilité de traiter cette information pour analyser, gérer, prévoir ou concevoir des événements dans des domaines aussi divers que la météo, la médecine, l'économie, la bureautique, etc.

Cette communication et ces traitements ne sont possibles qu'au travers de l'outil informatique. Cependant, toutes ces facultés résultent davantage de l'application d'un programme résidant sur l'ordinateur que de l'ordinateur lui-même. En fait, le programme est à l'ordinateur ce que l'esprit est à l'être humain.

Créer une application, c'est apporter de l'esprit à l'ordinateur. Pour que cet esprit donne sa pleine mesure, il est certes nécessaire de bien connaître le langage des ordinateurs, mais, surtout, il est indispensable de savoir programmer. La programmation est l'art d'analyser un problème afin d'en extraire la marche à suivre, l'algorithme susceptible de résoudre ce problème.

C'est pourquoi ce chapitre commence par aborder la notion d'algorithme. À partir d'un exemple tiré de la vie courante, nous déterminons les étapes essentielles à l'élaboration d'un programme (voir section « Construire un algorithme »). À la section suivante, « Qu'est-ce qu'un ordinateur ? », nous examinons le rôle et le fonctionnement de l'ordinateur dans le passage de l'algorithme au programme. Nous étudions ensuite, à travers un exemple simple, comment écrire un programme en Java et l'exécuter (voir section « Un premier programme en Java, ou comment parler à un ordinateur »). Enfin, nous décrivons, à la section « Le projet : Gestion d'un compte bancaire », le cahier des charges de l'application projet que le lecteur assidu peut réaliser en suivant les exercices décrits à la fin de chaque chapitre.

Construire un algorithme

Un ordinateur muni de l'application adéquate traite une information. Il sait calculer, compter, trier ou rechercher l'information, dans la mesure où un programmeur lui a donné les ordres à exécuter et la marche à suivre pour arriver au résultat.

Cette marche à suivre s'appelle un algorithme.

Déterminer l'algorithme, c'est trouver un cheminement de tâches à fournir à l'ordinateur pour qu'il les exécute. Voyons comment s'y prendre pour construire cette marche à suivre.

Copyright © 2016 Eyrolles.

Ne faire qu'une seule chose à la fois

Avant de réaliser une application concrète, telle que celle proposée en projet dans cet ouvrage, nécessairement complexe par la diversité des tâches qu'elle doit réaliser, simplifions-nous la tâche en ne cherchant à résoudre qu'un problème à la fois.

Considérons que créer une application, c'est décomposer cette dernière en plusieurs sous-applications qui, à leur tour, se décomposent en micro-applications, jusqu'à descendre au niveau le plus élémentaire. Cette démarche est appelée analyse descendante. Elle est le principe de base de toute construction algorithmique.

Pour bien comprendre cette démarche, penchons-nous sur un problème réel et simple à résoudre : comment faire un café chaud non sucré ?

Exemple : l'algorithme du café chaud

Construire un algorithme, c'est avant tout analyser l'énoncé du problème afin de définir l'ensemble des objets à manipuler pour obtenir un résultat.

Définition des objets manipulés

Analysons l'énoncé suivant :

```
Comment faire un café chaud non sucré ?
```

Chaque mot a son importance, et « non sucré » est aussi important que « café » ou « chaud ». Le terme « non sucré » implique qu'il n'est pas nécessaire de prendre du sucre ni une petite cuillère.

Notons que tous les ingrédients et ustensiles nécessaires ne sont pas cités dans l'énoncé. En particulier, nous ne savons pas si nous disposons d'une cafetière électrique ou non. Pour résoudre notre problème, nous devons prendre certaines décisions, et ces dernières vont avoir une influence sur l'allure générale de notre algorithme.

Supposons que, pour réaliser notre café, nous soyons en possession des ustensiles et ingrédients suivants :

```
café moulu
filtre
eau
pichet
cafetière électrique
tasse
électricité
table
```

En fixant la liste des ingrédients et des ustensiles, nous définissons un environnement, une base de travail. Nous sommes ainsi en mesure d'établir une liste de toutes les actions à mener pour résoudre le problème et de construire la marche à suivre permettant d'obtenir un café.

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

Liste des opérations

```
Verser l'eau dans la cafetière, le café dans la tasse, le café dans
le filtre.
Remplir le pichet d'eau.
Prendre du café moulu, une tasse, de l'eau, une cafetière électrique,
un filtre, le pichet de la cafetière.
Brancher, allumer ou éteindre la cafetière électrique.
Attendre que le café remplisse le pichet.
Poser la tasse, la cafetière sur la table, le filtre dans la
cafetière, le pichet dans la cafetière.
```

Cette énumération est une description de toutes les actions nécessaires à la réalisation d'un café chaud.

Chaque action est un fragment du problème donné et ne peut plus être découpée. Chaque action est élémentaire par rapport à l'environnement que nous nous sommes donné.

En définissant l'ensemble des actions possibles, nous créons un langage minimal qui nous permet de réaliser le café. Ce langage est composé de verbes (`Prendre`, `Poser`, `Verser`, `Faire`, `Attendre`, etc.) et d'objets (`Café moulu`, `Eau`, `Filtre`, `Tasse`, etc.).

La taille du langage, c'est-à-dire le nombre de mots qu'il renferme, est déterminée par l'environnement. Pour cet exemple, nous avons, en précisant les hypothèses, volontairement choisi un environnement restreint. Nous aurions pu décrire des tâches comme « prendre un contrat EDF » ou « planter une graine de café », mais elles ne sont pas utiles à notre objectif pédagogique.

Question Quelle serait la liste des opérations supplémentaires si l'on décidait de faire un café sucré ?

Réponse Les opérations seraient :

```
Prendre du sucre, une petite cuillère.
Poser le sucre dans la tasse, la cuillère dans la tasse.
```

Remarque Telle que nous l'avons décrite, la liste des opérations ne nous permet pas encore de faire un café chaud. En suivant cette liste, tout y est, mais dans le désordre. Pour réaliser ce fameux café, nous devons ordonner cette liste.

Ordonner la liste des opérations

```
1.  Prendre une cafetière électrique.
2.  Poser la cafetière sur la table.
3.  Prendre un filtre.
4.  Poser le filtre dans la cafetière.
5.  Prendre du café moulu.
6. Verser le café moulu dans le filtre.
```

Copyright © 2016 Eyrolles.

```
 7. Prendre le pichet de la cafetière.
 8. Remplir le pichet d'eau.
 9. Verser l'eau dans la cafetière.
10. Poser le pichet dans la cafetière.
11. Brancher la cafetière.
12. Allumer la cafetière.
13. Attendre que le café remplisse le pichet.
14. Prendre une tasse.
15. Poser la tasse sur la table.
16. Éteindre la cafetière.
17. Prendre le pichet de la cafetière.
18. Verser le café dans la tasse.
```

L'exécution de l'ensemble ordonné de ces tâches nous permet maintenant d'obtenir du café chaud non sucré.

Remarque L'ordre d'exécution de cette marche à suivre est important. En effet, si l'utilisateur réalise l'opération 12 (`Allumer la cafetière`) avant l'opération 9 (`Verser l'eau dans la cafetière`), le résultat est sensiblement différent. La marche à suivre ainsi désordonnée risque de détériorer la cafetière électrique.

Cet exemple tiré de la vie courante montre que, pour résoudre un problème, il est essentiel de définir les objets utilisés puis de trouver la suite logique de tous les ordres nécessaires à la résolution dudit problème.

Question Où placer les opérations supplémentaires, dans la liste ordonnée, pour faire un café sucré ?

Réponse Les opérations se placent à la fin de la liste précédente de la façon suivante :

```
19. Prendre du sucre.
20. Poser le sucre dans la tasse.
21. Prendre une petite cuillère.
22. Poser la cuillère dans la tasse.
```

Vers une méthode

La tâche consistant à décrire comment résoudre un problème n'est pas simple. Elle dépend en partie du niveau de difficulté du problème et réclame un savoir-faire : la façon de procéder pour découper un problème en actions élémentaires.

Pour aborder dans les meilleures conditions possibles la tâche difficile d'élaboration d'un algorithme, nous devons tout d'abord :

- Déterminer les objets utiles à la résolution du problème.

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

- Construire et ordonner la liste de toutes les actions nécessaires à cette résolution.

Pour cela, il est nécessaire :

- d'analyser en détail la tâche à résoudre ;
- de fractionner le problème en actions distinctes et élémentaires.

Ce fractionnement est réalisé en tenant compte du choix des hypothèses de travail. Ces hypothèses imposent un ensemble de contraintes, qui permettent de savoir si l'action décrite est élémentaire et ne peut plus être découpée.

Cela fait, nous avons construit un algorithme.

Passer de l'algorithme au programme

Pour construire un algorithme, nous avons défini des hypothèses de travail, c'est-à-dire supposé une base de connaissances minimales nécessaires à la résolution du problème. Ainsi, le fait de prendre l'hypothèse d'avoir du café moulu nous autorise à ne pas décrire l'ensemble des tâches précédant l'acquisition du café moulu. C'est donc la connaissance de l'environnement de travail qui détermine en grande partie la construction de l'algorithme.

Pour passer de l'algorithme au programme, le choix de l'environnement de travail n'est plus de notre ressort. Jusqu'à présent, nous avons supposé que l'exécutant était humain. Maintenant, notre exécutant est l'ordinateur. Pour écrire un programme, nous devons savoir ce dont est capable un ordinateur et connaître son fonctionnement de façon à établir les connaissances et capacités de cet exécutant.

Qu'est-ce qu'un ordinateur ?

Notre intention n'est pas de décrire en détail le fonctionnement de l'ordinateur et de ses composants mais d'en donner une image simplifiée.

Pour tenter de comprendre comment travaille l'ordinateur et, surtout, comment il se programme, nous allons schématiser à l'extrême ses mécanismes de fonctionnement.

Un ordinateur est composé de deux parties distinctes, la **mémoire centrale** et l'**unité centrale**.

La mémoire centrale permet de mémoriser toutes les informations nécessaires à l'exécution d'un programme. Ces informations correspondent à des **données** ou à des ordres à exécuter (**instructions**). Les ordres placés en mémoire sont effectués par l'unité centrale, la partie active de l'ordinateur.

Lorsqu'un ordinateur exécute un programme, son travail consiste en grande partie à gérer la mémoire, soit pour y lire une instruction, soit pour y stocker une information. En ce sens, nous pouvons voir l'ordinateur comme un robot qui sait agir en fonction des ordres qui lui sont fournis. Ces actions, en nombre limité, sont décrites ci-après.

Copyright © 2016 Eyrolles.

Déposer ou lire une information dans une case mémoire

La mémoire est formée d'éléments, ou cases, qui possèdent chacune un numéro (une adresse). Chaque case mémoire est en quelque sorte une boîte aux lettres pouvant contenir une information (une lettre). Pour y déposer cette information, l'ordinateur (le facteur) doit connaître l'adresse de la boîte. Lorsque le robot place une information dans une case mémoire, il mémorise l'adresse où se situe celle-ci afin de retrouver l'information en temps nécessaire.

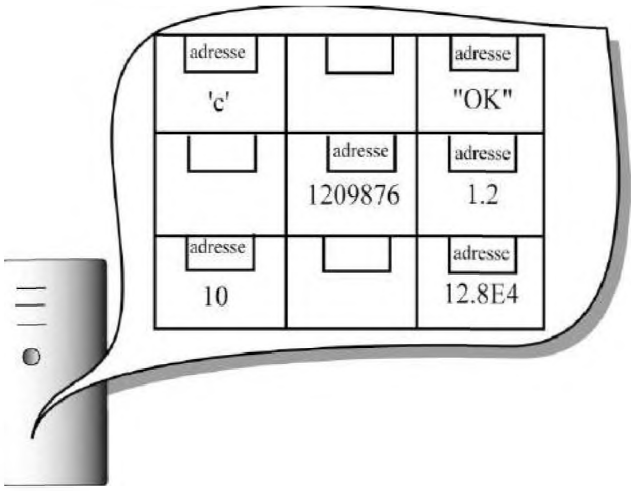


Figure I-1 *La mémoire de l'ordinateur est composée de cases possédant une adresse et pouvant contenir à tout moment une valeur.*

Le robot sait déposer une information dans une case, mais il ne sait pas la retirer (au sens de prendre un courrier déposé dans une boîte aux lettres). Lorsque le robot prend l'information déposée dans une case mémoire, il ne fait que la lire. En aucun cas il ne la retire ni ne l'efface. L'information lue reste toujours dans la case mémoire.

Remarque

Pour effacer une information d'une case mémoire, il est nécessaire de placer une nouvelle information dans cette même case. Ainsi, la nouvelle donnée remplace l'ancienne, et l'information précédente est détruite.

Exécuter des opérations simples telles que l'addition ou la soustraction

Le robot lit et exécute les opérations dans l'ordre où elles lui sont fournies. Pour faire une addition, il va chercher les valeurs à additionner dans les cases mémoire appropriées (stockées, par exemple, aux adresses `a` et `b`) et réalise ensuite l'opération demandée. Il enregistre alors le résultat de cette opération dans une case d'adresse `c`. De telles opérations sont décrites à l'aide d'ordres, appelés aussi **instructions**.

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

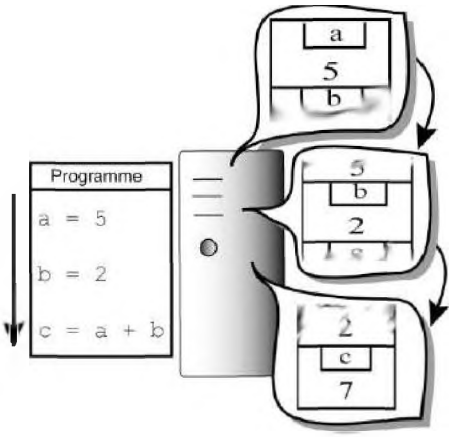


Figure I-2 *Le programme exécute les instructions dans l'ordre de leur apparition.*

Comparer des valeurs

Le robot est capable de comparer deux valeurs entre elles pour déterminer si l'une d'entre elles est plus grande, plus petite, égale ou différente de l'autre valeur. Grâce à la comparaison, le robot est capable de tester une condition et d'exécuter un ordre plutôt qu'un autre, en fonction du résultat du test.

La réalisation d'une comparaison ou d'un test fait que le robot ne peut plus exécuter les instructions dans leur ordre d'apparition. En effet, suivant le résultat du test, il doit rompre l'ordre de la marche à suivre, en sautant une ou plusieurs instructions. C'est pourquoi il existe des instructions particulières dites de **branchement**. Grâce à ce type d'instructions, le robot est à même non seulement de sauter des ordres mais aussi de revenir à un ensemble d'opérations afin de les répéter.

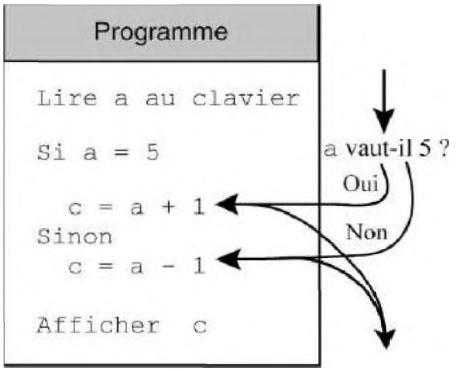


Figure I-3 *Suivant le résultat du test, l'ordinateur exécute l'une ou l'autre instruction en sautant celle qu'il ne doit pas exécuter.*

Copyright © 2016 Eyrolles.

Communiquer une information élémentaire

Un programme est essentiellement un outil qui traite l'information. Cette information est transmise à l'ordinateur par l'utilisateur. L'information est saisie par l'intermédiaire du clavier ou de la souris. Cette transmission de données à l'ordinateur est appelée communication d'entrée (*input* en anglais). On parle aussi de **saisie** ou encore de lecture de données.

Après traitement, le programme fournit un résultat à l'utilisateur, soit par l'intermédiaire de l'écran, soit sous forme de fichiers, que l'on peut ensuite imprimer. Il s'agit alors de communication de sortie (*output*) ou encore d'**affichage** ou d'**écriture** de données.

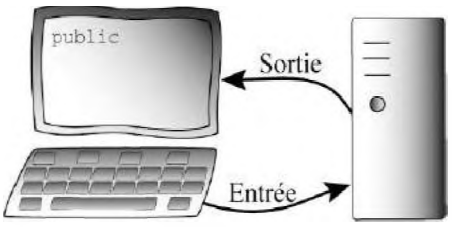


Figure I-4 *La saisie au clavier d'une valeur correspond à une opération d'entrée, et l'affichage d'un résultat à une opération de sortie.*

Coder l'information

De par la nature de ses composants électroniques, le robot ne perçoit que deux états : composant allumé et composant éteint. De cette perception découle le langage binaire, qui utilise par convention les deux symboles 0 (éteint) et 1 (allumé).

Ne connaissant que le 0 et le 1, l'ordinateur utilise un code pour représenter une information aussi simple qu'un nombre entier ou un caractère. Ce code est un programme, qui différencie chaque type d'information et transforme une information (donnée numérique ou alphabétique) en valeurs binaires. À l'inverse, ce programme sait aussi transformer un nombre binaire en valeur numérique ou alphabétique. Il existe autant de codes que de types d'informations. Cette différenciation du codage (en fonction de ce qui doit être représenté) introduit le concept de **type** de données.

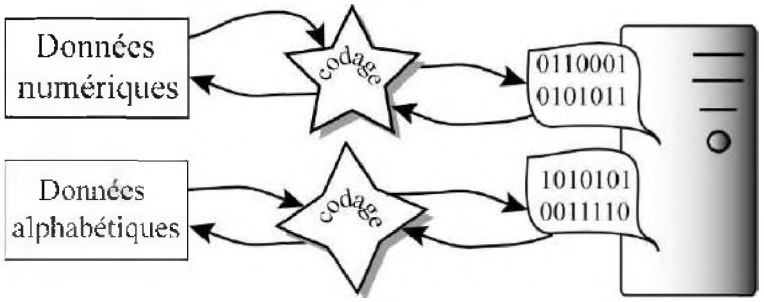


Figure I-5 *Toute information est codée en binaire. Il existe autant de codes que de types d'informations.*

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

Remarque Toute information fournie à l'ordinateur est, au bout du compte, codée en binaire. L'information peut être un simple nombre ou une instruction de programme.

Exemple

Pour additionner deux nombres, l'ordinateur fait appel aux trois éléments qui lui sont nécessaires pour réaliser cette opération. Ces éléments sont les suivants :

- Le code binaire représentant l'opération d'addition (par exemple 0101).
- L'adresse de la case mémoire où est stocké le premier nombre (par exemple 011101).
- L'adresse de la case mémoire où se trouve la deuxième valeur (par exemple 010101).

Pour finir, l'instruction d'addition de ces deux nombres s'écrit en assemblant les trois codes binaires (soit, dans notre exemple, 0101011101010101).

Remarque Le code binaire associé à chaque code d'opération (addition, test, etc.) n'est pas nécessairement identique d'un ordinateur à un autre. Ce code binaire est déterminé par le constructeur de l'ordinateur. De ce fait, une instruction telle que l'addition de deux nombres n'a pas le même code binaire d'une machine à une autre. Il existe donc, pour un même programme, un code binaire qui diffère suivant le type d'ordinateur utilisé.

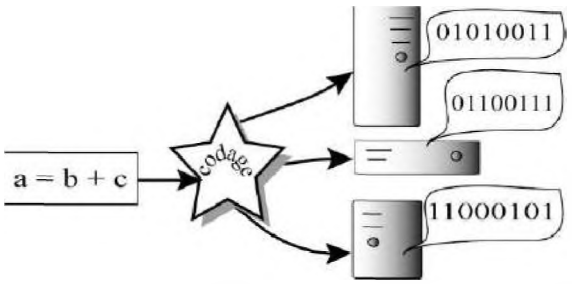


Figure I-6 *Pour un même programme, le code binaire diffère en fonction de l'ordinateur utilisé.*

L'ordinateur n'est qu'un exécutant

En pratique, le robot est très habile à réaliser l'ensemble des tâches énoncées ci-dessus. Il les exécute beaucoup plus rapidement qu'un être humain.

En revanche, le robot n'est pas doué d'intelligence. Il n'est ni capable de choisir une action plutôt qu'une autre, ni apte à exécuter de lui-même l'ensemble de ces actions. Pour qu'il puisse exécuter une instruction, il faut qu'un être humain détermine l'instruction la plus appropriée et lui donne l'ordre de l'exécuter.

Le robot est un exécutant capable de comprendre des ordres. Compte tenu de ses capacités limitées, les ordres ne peuvent pas lui être donnés dans le langage naturel propre à l'être humain.

Copyright © 2016 Eyrolles.

En effet, le robot ne comprend pas le sens des ordres qu'il exécute mais seulement leur forme. Chaque ordre doit être écrit avec des mots particuliers et une forme, ou syntaxe, préétablie. L'ensemble de ces mots constitue un langage informatique. Les langages C, C++, Pascal, Basic, Fortran, Cobol et Java sont des langages de programmation, constitués de mots et d'ordres dont la syntaxe diffère selon le langage.

Pour écrire un programme, il est nécessaire de connaître un de ces langages, de façon à traduire un algorithme en un programme composé d'ordres.

Un premier programme en Java, ou comment parler à un ordinateur

Pour créer une application, nous allons avoir à décrire une liste ordonnée d'opérations dans un langage compréhensible par l'ordinateur. La contrainte est de taille et se porte essentiellement sur la façon de définir et de représenter les objets nécessaires à la résolution du problème en fonction du langage de l'ordinateur.

Pour bien comprendre la difficulté du travail à accomplir, regardons comment faire calculer à un ordinateur la circonférence d'un cercle de rayon quelconque.

Calcul de la circonférence d'un cercle

L'exercice consiste à calculer le périmètre d'un cercle de rayon quelconque. Nous supposons que l'utilisateur emploie le clavier pour transmettre au programme la valeur du rayon.

Définition des objets manipulés

Pour calculer la circonférence du cercle, l'ordinateur a besoin de stocker dans ses cases mémoire la valeur du rayon ainsi que celle du périmètre. Les objets à manipuler sont deux valeurs numériques appartenant à l'ensemble des réels $\mathbb{R}$. Nous appelons `lePerimetre` la valeur correspondant au périmètre et `unRayon` la valeur du rayon.

La liste des opérations

La circonférence d'un cercle est calculée à partir de la formule :

$$\texttt{LePerimetre} = 2 \times \pi \times \texttt{unRayon}.$$

La valeur du rayon est fournie par l'utilisateur à l'aide du clavier. Elle n'est donc pas connue au moment de l'écriture du programme. En conséquence, il est nécessaire d'écrire l'ordre (instruction) de saisie au clavier avant de calculer la circonférence.

La liste des opérations est la suivante :

```
1. Réserver deux cases mémoire pour y stocker les valeurs
   ➥correspondant au rayon (unRayon) et au périmètre (lePerimetre).
2. Demander à l'utilisateur de saisir la valeur du rayon au clavier
   ➥et la placer dans la case mémoire associée.
3. Connaissant la valeur du rayon, calculer la circonférence.
4. Afficher le résultat.
```

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

La valeur du rayon puis, après calcul, celle de la circonférence sont les données principales de ce programme. L'ordinateur doit les stocker en mémoire pour les utiliser.

L'opération 1 consiste à donner un nom aux cases mémoire qui vont servir à stocker ces données. Lors de cette opération, appelée **déclaration de variables**, l'ordinateur réserve une case mémoire pour chaque nom de variable défini. Ici, ces variables ont pour nom P et R. Au cours de cette réservation d'emplacements mémoire, l'ordinateur associe le nom de la variable et l'adresse réelle de la case mémoire.

Remarque

Pour le programmeur, le nom et l'adresse d'une case ne font qu'un, car il ne manipule les variables que par leur nom, alors que l'ordinateur travaille avec leur adresse. En donnant un nom à une case, l'être humain sait facilement identifier les objets qu'il manipule, alors qu'il lui serait pénible de manipuler les adresses binaires correspondantes. Inversement, en associant un nom à une adresse codée en binaire, l'ordinateur peut véritablement manipuler ces objets.

L'opération 2 permet de saisir au clavier la valeur du rayon. Pour que l'utilisateur non initié sache à quoi correspond la valeur saisie, il est nécessaire, avant de procéder à cette saisie, d'afficher un message explicatif à l'écran. L'opération 2 se décompose en deux instructions élémentaires, à savoir :

```
Afficher un message demandant à l'utilisateur du programme de saisir
➥une valeur pour le rayon.
Une fois la valeur saisie par l'utilisateur, la placer dans sa case
➥mémoire.
```

Les opérations 3 et 4 sont des actions élémentaires directement traduisibles en langage informatique.

La traduction en Java

Une application, ou programme, ne s'écrit pas en une seule fois. Nous verrons à la lecture de cet ouvrage que programmer c'est toujours décomposer une difficulté en différentes tâches plus aisées à réaliser. Cette décomposition s'applique aussi bien pour construire un algorithme que pour l'écriture du programme lui-même.

D'une manière générale, la meilleure façon de procéder pour fabriquer un programme revient à écrire une première ébauche et à la tester. De ces tests, il ressort des fautes à corriger et, surtout, de nouvelles idées. Le programme final consiste en l'assemblage de toutes ces corrections et de ces améliorations.

Pour traduire la marche à suivre définie précédemment selon les règles de syntaxe du langage Java, nous allons utiliser cette même démarche. Nous nous intéresserons, dans un premier temps, à la traduction du cœur du programme (opérations 1 à 4 décrites à la section précédente).

Copyright © 2016 Eyrolles.

Nous verrons pour finir comment insérer l'ensemble de ces instructions dans une structure de programme Java.

- L'opération 1 consiste à déclarer les variables utilisées pour le calcul de la circonférence. Cette opération se traduit par l'instruction :

```
double unRayon, lePerimetre ;
```

Par cette instruction, le programme demande à l'ordinateur de réserver deux cases mémoire, nommées `unRayon` et `lePerimetre`, pour y stocker les valeurs du rayon et de la circonférence. Le mot réservé `double` permet de préciser que les valeurs numériques sont réelles « avec une double précision », c'est-à-dire avec une précision pouvant aller jusqu'à 17 chiffres après la virgule.

Pour en savoir plus Pour plus d'informations sur la définition des types de variables, reportez-vous au chapitre 1, « Stocker une information ».

- L'opération 2 s'effectue en deux temps :
  1. Afficher un message demandant à l'utilisateur de saisir une valeur. Cette opération se traduit par l'instruction :

```
System.out.print("Valeur du rayon : ") ;
```

`System.out.print()` est ce que l'on appelle un programme, ou une fonction, prédéfini par le langage Java. Ce programme permet d'écrire à l'écran le message spécifié à l'intérieur des parenthèses. Le message affiché est ici un fragment de texte, appelé, dans le jargon informatique, une chaîne de caractères. Pour que l'ordinateur comprenne que la chaîne de caractères n'est pas un nom de variable mais un texte à afficher, il faut placer entre guillemets (`" "`) tous les caractères composant la chaîne.

  2. Saisir et stocker la valeur demandée en mémoire. Pour ce faire, nous devons écrire les instructions suivantes :

```
Scanner lectureClavier = new Scanner(System.in);
unRayon = lectureClavier.nextDouble();
```

`Scanner` est un outil (une classe) proposé à partir de la version 1.5 de Java qui permet à l'utilisateur de communiquer une valeur numérique au programme par l'intermédiaire du clavier. Cet outil utilise des fonctions (par exemple `nextDouble()`) qui donnent l'ordre à l'ordinateur d'attendre la saisie d'une valeur (de double précision, pour notre exemple). La saisie est effective lorsque l'utilisateur valide sa réponse en appuyant sur la touche `Entrée` du clavier.

Une fois la valeur saisie, celle-ci est placée dans la case mémoire nommée `unRayon` grâce au signe =.

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

Pour en savoir plus Pour plus de précisions sur les deux méthodes `System.out.print()` et lecture `Clavier.nextDouble()` reportez-vous au chapitre 2, « Communiquer une information ». Pour le signe égal (=), voir le chapitre 1, « Stocker une information ». La notion de classe et l'opérateur `new` sont étudiés au chapitre 7, « Les classes et les objets ».

- L'opération 3 permet de calculer la valeur de la circonférence. Elle se traduit de la façon suivante :

```
lePerimetre = 2 * Math.PI * unRayon ;
```

Le signe * est le symbole qui caractérise l'opération de multiplication. `Math.PI` est le terme qui représente la valeur numérique du nombre π avec une précision de 17 chiffres après la virgule. Le mot-clé `Math` désigne la boîte à outils composée de fonctions mathématiques accompagnant le langage Java. Cette bibliothèque contient, outre des constantes telles que `p`, des fonctions standards, comme `sqrt()` (racine carrée) ou `sin()` (sinus). Une fois les opérations de multiplication effectuées, la valeur calculée est placée dans la variable `lePerimetre` grâce au signe =.

- La dernière opération (4) de notre programme a pour rôle d'afficher le résultat du calcul précédent. Cet affichage est réalisé grâce à l'instruction :

```
System.out.print("Le cercle de rayon " + unRayon +
                 " a pour perimetre : " + lePerimetre);
```

Ce deuxième appel à la fonction `System.out.print()` est plus complexe que le premier. Il mélange l'affichage de chaînes de caractères (texte entre guillemets) et de contenu de variables.

Si les caractères `R` et `P` ne sont pas placés entre guillemets, c'est pour que l'ordinateur les interprète non pas comme des caractères à afficher mais comme les variables qui ont été déclarées en début de programme. De ce fait, il affiche le contenu des variables et non les lettres `R` et `P`.

Les signes +, qui apparaissent dans l'expression `"Le cercle de rayon " + unRayon + " a pour perimetre : " + lePerimetre`, indiquent que chaque élément du message doit être affiché en le collant aux autres : d'abord la chaîne de caractères `"Le cercle de rayon "`, puis la valeur de `unRayon`, puis la chaîne `"a pour périmètre :"` et, pour finir, la valeur de `lePerimetre`.

En résumé, la partie centrale du programme contient les cinq instructions suivantes :

```
double unRayon, lePerimetre ;
Scanner lectureClavier = new Scanner(System.in);
System.out.print("Valeur du rayon : ") ;
```

Copyright © 2016 Eyrolles.

```
unRayon = lectureClavier.nextDouble();
lePerimetre = 2 * Math.PI * unRayon ;
System.out.print("Le cercle de rayon " + unRayon +
                 " a pour perimetre : " + lePerimetre );
```

Pour améliorer la lisibilité du programme, il est possible d'insérer dans le programme, des commentaires, comme suit :

```
// 1. Déclarer les variables
double unRayon, lePerimetre ;
Scanner lectureClavier = new Scanner(System.in);
// 2.a Afficher le message "Valeur du rayon : " à l'écran
System.out.print("Valeur du rayon : ") ;
// 2.b Lire au clavier une valeur, placer cette valeur dans la
   ➥variable unRayon
unRayon = lectureClavier.nextDouble();
// 3. Calculer la circonférence en utilisant la formule consacrée
lePerimetre = 2 * Math.PI * unRayon ;
// 4. Afficher le résultat
System.out.print("Le cercle de rayon " + unRayon +
                 " a pour perimetre : " + lePerimetre );
```

Les lignes du programme qui débutent par les signes // sont considérées par l'ordinateur non pas comme des ordres à exécuter mais comme des lignes de commentaire. Elles permettent d'expliquer en langage naturel ce que réalise l'instruction associée.

Écrites de la sorte, ces instructions constituent le cœur de notre programme. Elles ne peuvent cependant pas encore être interprétées correctement par l'ordinateur. En effet, celui-ci exécute les instructions d'un programme dans l'ordre de leur arrivée. Une application doit donc être constituée d'une instruction qui caractérise le début du programme. Pour ce faire, nous devons écrire notre programme ainsi :

```
public static void main(String [] arg)
{
  // 1. Déclarer les variables
  double unRayon, lePerimetre ;
  Scanner lectureClavier = new Scanner(System.in);
  // 2.a Afficher le message "Valeur du rayon : " à l'écran
  System.out.print("Valeur du rayon : ") ;
  // 2.b Lire au clavier une valeur, placer cette valeur
  // dans la variable unRayon
  unRayon = lectureClavier.nextDouble();
  // 3. Calculer la circonférence en utilisant la formule consacrée
  lePerimetre = 2 * Math.PI * unRayon ;
```

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

```
    // 4. Afficher le résultat
    System.out.print("Le cercle de rayon " + unRayon +
                     " a pour perimetre : " + lePerimetre );
} // Fin de la fonction main()
```

La ligne `public static void main(String [] arg)` est l'instruction qui permet d'indiquer à l'ordinateur le début du programme. Ce début est identifié par ce que l'on appelle la fonction `main()`, c'est-à-dire la fonction principale du programme. De cette façon, lorsque le programme est exécuté, l'ordinateur recherche le mot-clé `main`. Une fois ce mot-clé trouvé, l'ordinateur exécute une à une chaque instruction constituant la fonction.

Pour en savoir plus Les autres mots-clés, tels que `public`, `static` ou `void`, déterminent certaines caractéristiques de la fonction `main()`. Ces mots-clés, obligatoirement placés et écrits dans cet ordre, sont expliqués au fur et à mesure de leur apparition dans le livre et plus particulièrement à la section « Quelques techniques utiles » du chapitre 9, « Collectionner un nombre fixe d'objets ».

Pour finir, nous devons insérer la fonction `main()` dans ce qui est appelé une classe Java. En programmation objet, un programme n'est exécutable que s'il est défini à l'intérieur d'une classe. Une classe est une entité interprétée par l'ordinateur comme étant une unité de programme, qu'il peut exécuter dès qu'un utilisateur le souhaite.

Aucun programme ne peut être écrit en dehors d'une classe. Nous devons donc placer la fonction `main()` à l'intérieur d'une classe définie par l'instruction `public class Cercle {}`, comme suit :

```
public class Cercle
{
 public static void main(String [] arg)
 {
   // 1. Déclarer les variables
   double unRayon, lePerimetre ;
   Scanner lectureClavier = new Scanner(System.in);
   // 2.a Afficher le message "Valeur du rayon : " à l'écran
   System.out.print("Valeur du rayon : ") ;
   // 2.b Lire au clavier une valeur, placer cette valeur dans
   // la variable unRayon
   unRayon = lectureClavier.nextDouble();
   // 3. Calculer la circonférence en utilisant la formule consacrée
   lePerimetre = 2 * Math.PI * unRayon ;
```

Copyright © 2016 Eyrolles.

```
    // 4. Afficher le résultat
    System.out.print("Le cercle de rayon " + unRayon +
                     " a pour perimetre : " + lePerimetre );
  }
} // Fin de la classe Cercle
```

Nous obtenons ainsi le programme dans son intégralité. La ligne `public class Cercle` permet de définir une classe. Puisque notre programme effectue des opérations sur un cercle, nous avons choisi d'appeler cette classe `Cercle`. Nous aurions pu lui donner un tout autre nom, comme `Rond` ou `Exemple`. Ainsi définie, la classe `Cercle` devient un programme à part entière.

Pour finir, il convient de débuter le programme par l'instruction :

```
import java.util.*;
```

Cette instruction est obligatoire lorsqu'on utilise la classe `Scanner`. Elle indique au compilateur qu'il doit charger les classes (et notamment la classe `Scanner`) enregistrées dans la boîte à outils (*package*) `java.util` avant de commencer la phase de compilation. Si vous omettez cette instruction, le compilateur Java signale une erreur du type `cannot find symbol class Scanner`.

Remarque

Les classes standards de Java sont regroupées par paquetage (en anglais *package*). Par exemple toutes les classes relatives au traitement de texte son regroupées dans le paquetage `java.text`, le paquetage `java.awt` fournit quant à lui toutes les classes correspondant à la gestion des interfaces graphiques.

Pour en savoir plus

Pour voir le résultat de l'exécution de ce programme, reportez-vous à la section « Exemple sur plate-forme Unix », ci-après.

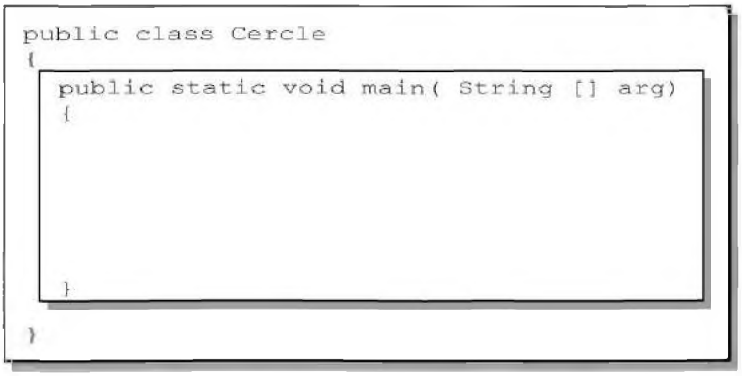


Figure I-7 *Un programme Java est constitué de deux blocs encastrés. Le premier bloc représente la classe associée au programme, tandis que le second détermine la fonction principale.*

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

En observant la figure 7, nous constatons que ce programme, de même que tous ceux à venir, est constitué de deux blocs encastrés définis par les deux lignes `public class Cercle{}` et `public static void main(String [] arg){}`.

Ces deux blocs constituent la charpente principale et nécessaire à tout programme écrit avec le langage Java. Cet exemple montre en outre que les mots réservés par le langage Java sont nombreux et variés et qu'ils constituent une partie du langage Java.

Si la syntaxe, c'est-à-dire la forme, de ces instructions peut paraître étrange de prime abord, nous verrons à la lecture de cet ouvrage que leur emploi obéit à des règles strictes. En apprenant ces règles et en les appliquant, vous pourrez vous initier aux techniques de construction d'un programme, qui reviennent à décomposer un problème en actions élémentaires puis à traduire celles-ci à l'aide du langage Java.

Question Dans la classe `Cercle`, quelle instruction faut-il modifier pour calculer non plus le périmètre, mais la surface d'un cercle ?

Réponse La surface d'un cercle est obtenue par la formule : `Surface = π × Rayon × Rayon`.

Il suffit donc de modifier l'instruction de déclaration :

```
// Déclaration des variables
  double unRayon, lePerimetre ;
```

en :

```
// Déclaration des variables
  double unRayon, laSurface ;
```

Puis de remplacer l'instruction :

```
// Calculer la circonférence en utilisant la formule consacrée
  lePerimetre = 2*Math.PI*unRayon ;
```

en :

```
// Calculer la surface en utilisant la formule consacrée
  laSurface = Math.PI*unRayon*unRayon ;
```

Et pour finir remplacer l'instruction :

```
    // . Afficher le résultat
    System.out.print("Le cercle de rayon " + unRayon +
                     " a pour perimetre : " + lePerimetre );
```

en :

```
    // . Afficher le résultat
    System.out.print("Le cercle de rayon " + unRayon +
                     " a pour surface : " + laSurface );
```

Copyright © 2016 Eyrolles.

Exécuter un programme

Nous avons écrit un programme constitué d'ordres, dont la syntaxe obéit à des règles strictes. Pour obtenir le résultat des calculs décrits dans le programme, nous devons le faire lire par l'ordinateur, c'est-à-dire l'exécuter.

Pour cela, nous devons traduire le programme en langage « compréhensible » par l'ordinateur. En effet, nous l'avons vu, l'ordinateur ne comprend qu'un seul langage, le langage binaire.

Compiler, ou traduire en langage machine

Cette traduction du code source (le programme écrit en langage informatique) en code machine exécutable (le code binaire) est réalisée par un programme appelé **compilateur**. L'opération de compilation consiste à lancer un programme qui lit chaque instruction du code source et vérifie si celles-ci ont une syntaxe correcte. S'il n'y a pas d'erreur, le compilateur crée un code binaire directement exécutable par l'ordinateur.

Il existe autant de compilateurs que de langages. Un programme écrit en langage Pascal est traduit en binaire à l'aide d'un compilateur Pascal, et un programme écrit en Java est compilé par un compilateur Java. Le compilateur Java ne travaille pas tout à fait comme un compilateur classique, traduisant un code source en code exécutable. Pour mieux comprendre cette différence, voyons son fonctionnement et comment l'utiliser.

Compiler un programme écrit en Java

L'objectif premier de J. Gosling, le créateur du langage Java, a été de réaliser un langage indépendant de l'ordinateur. Dans cette optique, un programme écrit sur PC, par exemple, doit pouvoir s'exécuter sur un PC (de type IBM), un Macintosh (Apple) ou une station Unix (de type Sun), et ce sans réécriture ni compilation du code source.

Or, le code binaire est spécifique à chaque machine, comme nous l'avons vu à la section « Coder l'information ». Il est impossible de faire tourner un même programme source d'une machine à une autre sans le compiler à nouveau. En effet, lors de la nouvelle compilation, des erreurs apparaissent, dues aux différences de matériel informatique. Pour corriger cet inconvénient majeur, l'idée de J.Gosling a été de créer un code intermédiaire entre le code source et le code binaire. Ce code intermédiaire est appelé pseudo-code, ou encore byte code.

En effet, en créant un pseudo-code, identique pour tous les ordinateurs, il est possible d'exécuter ce code sur différentes machines, sans avoir à le recompiler. Cette exécution est réalisée par un programme spécifique de la machine utilisée, qui interprète et exécute le pseudo-code, compte tenu des ressources propres de l'ordinateur.

Ce programme s'appelle un interpréteur Java. Il en existe autant que de types d'ordinateurs (plates-formes). L'ensemble des ces interpréteurs constitue ce que l'on appelle la machine virtuelle Java, ou JVM (*Java Virtual Machine*).

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

Le compilateur Java ne crée pas de code binaire, à la différence des autres compilateurs, tels que les compilateurs C ou C++. Il fabrique un pseudo-code, qui est ensuite interprété par un programme spécifique de l'ordinateur. Ce dernier programme transforme le pseudo-code en code directement exécutable par l'ordinateur choisi. L'avantage d'un tel système est que le développeur d'applications est certain de créer des programmes totalement compatibles avec les différents ordinateurs du marché sans avoir à réécrire une partie du code.

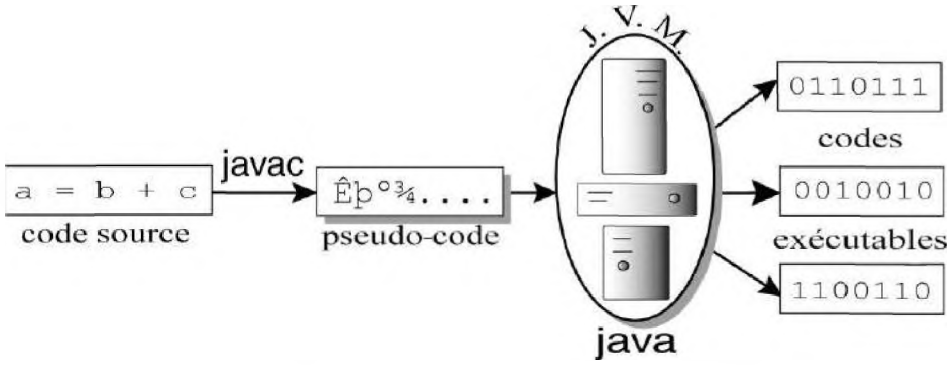


Figure I-8 *Le compilateur (javac) transforme le code source en pseudo-code. Ce dernier est exécuté grâce à un interpréteur (java) spécifique à chaque type de machine. L'ensemble des interpréteurs constitue la JVM.*

Le kit de développement Java (JDK)

Le tout premier compilateur Java a été écrit par J. Gosling à l'initiative de Sun, le constructeur de stations de travail sous Unix, au début des années 1990.

Aujourd'hui, le compilateur Java est téléchargeable depuis le site Internet de Sun. Il est fourni avec le kit de développement Java (JDK *Java Development Kit* ou encore SDK *Standard Development Kit*). Cet environnement est disponible sur les ordinateurs de type Solaris, PC sous Windows, Linux et Mac OS.

Pour en savoir plus Toutes les informations nécessaires à l'installation du JDK sur votre machine (Windows et Linux), sont fournies dans l'annexe « Guide d'installations », section « Installation d'un environnement de développement ».

Le JDK est ce que l'on appelle une boîte à outils de développement d'applications. Cela revient à dire qu'il est constitué d'outils, ou programmes, que l'on utilise sous forme de commande, ou ordre. Pour transmettre une commande à un ordinateur, le programmeur doit saisir le nom de cette commande au clavier, dans une fenêtre spécifique du type d'ordinateur utilisé. Les deux principales commandes à connaître pour cet ouvrage sont les commandes de compilation (`javac`) et d'exécution (`java`).

Copyright © 2016 Eyrolles.

Exemple sur plate-forme Unix

La marche à suivre est la suivante :

1. Entrez le programme qui calcule la circonférence d'un cercle (exemple donné à la section « Écrire un programme ») à l'aide d'un éditeur de texte, c'est-à-dire un logiciel permettant de saisir du texte au clavier. Les éditeurs de texte les plus couramment utilisés sous Unix, sont `vi` et `emacs`.
2. Sauvegardez votre programme en choisissant comme nom de fichier celui qui suit les termes `public class`. Pour notre exemple, nous avons écrit `public class Cercle`. Le fichier est donc à sauvegarder sous le nom `Cercle.java`.
3. Lancez l'ordre de compilation en saisissant sous Unix la commande :

   ```
   javac Cercle.java
   ```

   La compilation est lancée. Le compilateur exécute sa tâche et compile le fichier `Cercle.java`. Au final, si aucune erreur n'est détectée, le compilateur crée un nouveau fichier, appelé `Cercle.class`. Ce fichier correspond au pseudo-code relatif au programme compilé.

Remarque

Il convient de respecter la façon dont est nommé le fichier, orthographe, majuscule et minuscule compris. L'utilisation de l'extension `.java` est obligatoire.

4. Exécutez le programme en lançant la commande :

   ```
   java Cercle
   ```

   La commande `java` lance le programme, qui interprète le pseudo-code créé à l'étape précédente. Ce programme traduit le pseudo-code dans le code binaire conforme à la machine sur laquelle il est lancé. Après exécution, le résultat obtenu à l'écran est :

   ```
   Valeur du rayon: 5
   Le cercle de rayon 5 a pour perimetre: 31.41592653589793
   ```

où `5` est une valeur entrée au clavier par l'utilisateur.

Remarque

Notez que la commande de lancement du programme ne demande aucune extension, seule la commande `java` suivie du nom du fichier suffit.

Pour exécuter un programme, les deux étapes suivantes sont nécessaires :

- La compilation du programme à l'aide de la commande `javac`, suivie du nom du programme. Une fois la commande réalisée, le pseudo-code est créé et enregistré dans un fichier, dont le nom correspond au nom du programme suivi de l'extension `".class"`.

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

- L'exécution du programme en appelant l'interpréteur au moyen de la commande `java`, suivie du nom du programme (sans extension). Cette commande interprète le fichier `".class"` créé à l'étape précédente et exécute le programme.

Question

En supposant que le programme qui calcule la surface d'un cercle s'apppelle `SurfaceCercle` :

1. Quel est le nom du fichier associé à ce programme ?
2. Quelle est la commande pour le compiler ?
3. Quelle est la commande pour l'exécuter ?

Réponse

1. Le nom du fichier est `SurfaceCercle.java`.
2. La commande javac `SurfaceCercle.java` permet de compiler le programme.
3. Lorsqu'il n'y a plus d'erreurs signalées par le compilateur, la commande java `SurfaceCercle` lance l'exécution du programme.

Les environnements de développement

JDK fournit un ensemble de commandes pour compiler et interpréter. C'est un environnement courant et facile d'emploi dans le monde Unix. Il l'est beaucoup moins, en revanche, sous Windows. En effet, l'écriture d'une commande telle que donner l'ordre de compiler un programme ne peut se réaliser qu'en ouvrant une fenêtre « Commandes ».

Un certain nombre d'environnements de programmation permettent cependant d'écrire, de compiler puis d'exécuter de façon conviviale un programme Java. Citons, à titre d'exemples les environnements de travail, tels que le logiciel Eclipse (Eclipse.org), sous Windows ou Linux, ou Visual Café (Symantec), Project Builder (Apple) sur Macintosh et NetBeans sous Windows, Linux et Mac OS.

Ces logiciels offrent, sous forme d'interface graphique conviviale, un ensemble d'outils de développement d'applications. Les outils les plus utilisés sont, en général, les suivants :

- L'éditeur de texte pour écrire le programme.
- Les menus et boîtes à outils, pour lancer la compilation et l'exécution.
- La fenêtre de compilation, qui affiche les éventuelles erreurs de syntaxe.
- La fenêtre d'exécution, qui affiche les messages et résultats du programme en cours.
- La fenêtre qui visualise les projets en cours, dans le cas d'un programme défini à partir de plusieurs fichiers différents.

Pour en savoir plus

Pour plus d'informations, ou si vous souhaitez travailler avec l'interface NetBeans, vous trouverez toutes les informations nécessaires à son installation et à son utilisation dans l'annexe « Guide d'installations », sections « Installation d'un environnement de développement » et « Développer avec NetBeans ».

Copyright © 2016 Eyrolles.

Le projet : Gestion d'un compte bancaire

Pour vous permettre de mieux maîtriser les différentes notions abordées dans cet ouvrage, nous vous proposons de construire une application plus élaborée que les simples exercices appliqués donnés en fin de chapitre.

Dans ce projet, nous avons volontairement évité l'emploi d'interfaces graphiques. Bien qu'attrayantes, ces dernières sont difficiles à maîtriser pour des débutants en programmation. Le projet consiste à bâtir une application autour du concept de menu interactif. À l'heure du tout graphique, il n'est pas vain d'apprendre à écrire des menus « texte ». Le fait de passer par cet apprentissage permet d'appréhender toutes les notions fondamentales de la programmation, sans avoir à s'évertuer à étudier la syntaxe de toutes les méthodes de la bibliothèque graphique Java.

Cahier des charges

Il s'agit d'écrire une application interactive qui permet de gérer l'ensemble des comptes bancaires d'une personne. Les fonctionnalités fournies par le programme de gestion de comptes bancaires sont les suivantes :

```
Création, Suppression d'un compte
Affichage d'un compte donné
Saisie d'une ligne comptable pour un compte donné
Calcul de statistiques
Sauvegarde des données (n° de compte, lignes comptables)
```

Niveau 1 : programme interactif sous forme de choix dans un menu

L'exécution du programme affiche le menu suivant :

```
1.  Créer un compte
2.  Afficher un compte
3.  Créer une ligne comptable
4.  Sortir
5.  De l'aide

Votre choix :
```

L'utilisateur choisit une valeur pour exécuter l'opération souhaitée (les zones grisées correspondent à des valeurs choisies par l'utilisateur).

- Si l'utilisateur choisit l'option `1`, les informations à fournir concernent :

```
Le type du compte [Types possibles : Compte courant, joint,
épargne] :
Le numéro du compte :
La première valeur créditée :
Le taux de placement dans le cas d'un compte épargne :
```

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

- Si l'utilisateur choisit l'option `2`, le programme affiche les caractéristiques d'un compte (type, valeur courante, taux), ainsi que les dix dernières opérations comptables dans l'ordre des dates où ont été effectuées les opérations.
- Pour l'option `3`, il s'agit de fournir des informations pour créer une ligne comptable. Ces informations sont les suivantes :

```
Le numéro du compte concerné (avec vérification de son existence) :
La somme à créditer (valeur positive) ou à débiter (valeur néga-
tive) :
La date de l'opération :
Le motif de l'achat ou de la vente [thèmes possibles : Salaire,
Loyer, Alimentation, Divers] :
Le mode de paiement[Types possibles : CB, n° du Chèque, Virement]:
```

- L'option `4. Sortir` du menu général permet de sortir du programme.
- L'option `5. Aide` du menu général affiche une information relative à chaque option du menu.

Niveau 2 : structure de données optimisée en termes d'utilisation de la mémoire

- Le programme doit pouvoir gérer autant de comptes que nécessaire. Pour chaque compte, le nombre d'opérations comptables doit être infini et est donc indéterminable au moment de l'écriture du programme.
- En conséquence, la réservation des cases mémoire ne peut pas être réalisée de façon définitive en tout début de programme. À chaque ligne comptable et à chaque nouveau compte créé, le programme doit être capable de réserver lui-même le nombre suffisant d'emplacements mémoire pour la bonne marche du programme. Lorsque le programme gère lui-même la réservation des emplacements mémoire, on dit qu'il gère sa mémoire de manière dynamique.
- L'option permettant la suppression d'un compte est dépendante de la façon dont est stockée l'information. Cette option ne peut être abordée avant d'avoir choisi le mode de gestion des emplacements mémoire.
- L'option `5. Sortir` du menu général doit contrôler la sauvegarde de l'information. Les données sont sauvegardées sur disque sous forme d'un fichier portant le nom `compte.dat`.

Niveau 3 : s'initier aux graphiques

Un nouveau choix est ajouté à l'option `2. Afficher un compte` du menu général : il s'agit d'afficher les statistiques pour un compte donné sous différentes formes graphiques (histogramme, camembert, etc.).

Copyright © 2016 Eyrolles.

Niveau 4 : s'initier à la programmation d'interfaces graphiques

L'objectif est de rendre l'application plus conviviale en la dotant d'une interface graphique ergonomique.

Les actions à réaliser ne sont plus proposées sous la forme de menus textuels mais à l'aide de fenêtres composées de champs de saisie, de listes déroulantes... Ainsi, la première fenêtre de l'application propose de créer, d'éditer un compte ou d'ajouter une ligne comptable (voir figure I-9).

Selon le choix réalisé par l'utilisateur, différentes fenêtres s'affichent ensuite. Elles présentent de nouveaux formulaires permettant la saisie des informations nécessaires à la création d'un compte ou à une ligne comptable.

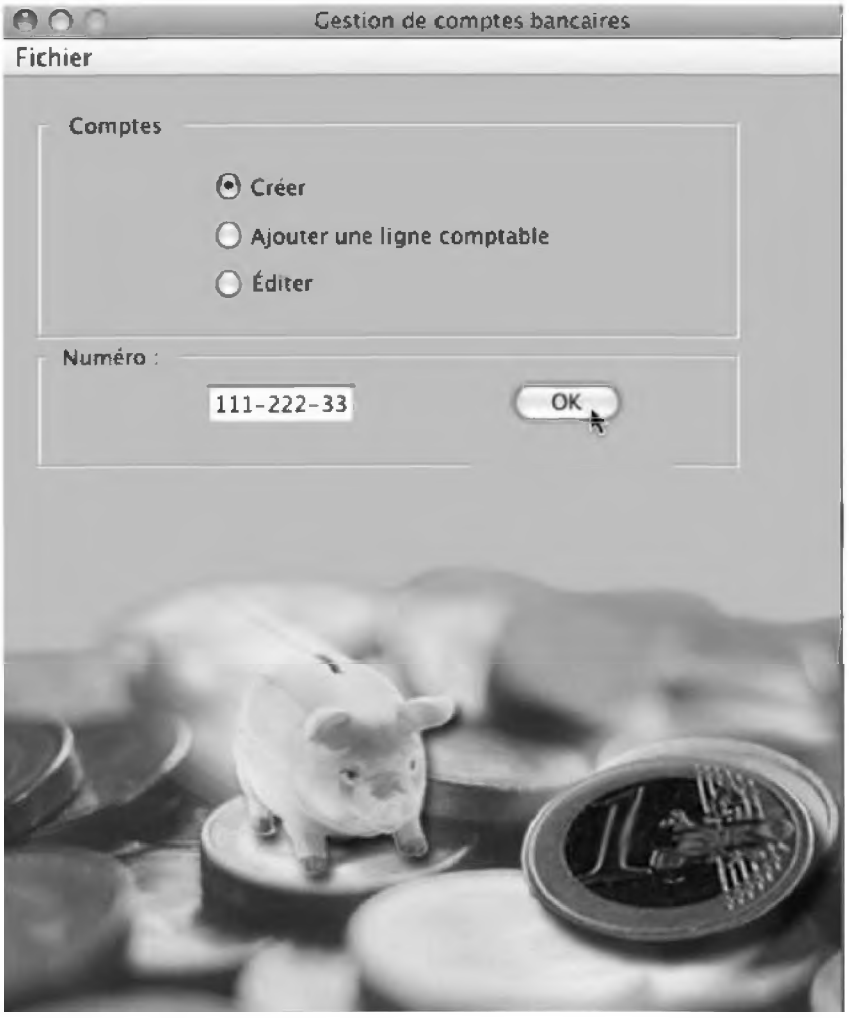


Figure I-9 *Panneau d'entrée de l'application muni d'une interface graphique.*

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

Les objets manipulés

Un compte bancaire est défini par un ensemble de données :

- un numéro du compte ;
- un type de compte (courant, épargne, joint, etc.) ;
- des lignes comptables possédant chacune une valeur, une date, un thème et un moyen de paiement.

Ces données peuvent être représentées de la façon suivante :

Données	Exemple	Type de l'objet
Numéro du compte	4010.205.530	Caractère
Type du compte	Courant	Caractère
Valeur	– 1 520,30	Numérique
Date	04 03 1978	Date
Thème	Loyer	Caractère
Moyen de paiement	CB	Caractère

Nous verrons, au chapitre 1, « Stocker une information », puis tout au long du chapitre 7, « Les classes et les objets », comment définir et représenter les objets utiles et nécessaires à la réalisation de cette application.

La liste des ordres

Pour créer une application de gestion de comptes bancaires, nous devons décomposer l'ensemble de ses fonctionnalités en tâches élémentaires. Pour ce faire, nous partageons l'application en trois niveaux, de difficulté croissante. Les niveaux 1 et 2 doivent être abordés dans cet ordre et sont nécessaires à la réalisation du niveau 3. La mise en œuvre du niveau 1 permet de réaliser les actions suivantes :

- construire un menu (voir les chapitres 2, « Communiquer une information », et 3, « Faire des choix ») ;
- créer des comptes différents ou saisir plusieurs lignes comptables (voir les chapitres 4, « Faire des répétitions », et 9, « Collectionner un nombre fixe d'objets ») ;
- définir les comptes et les lignes comptables comme des objets informatiques, au sens de la programmation objet (voir les chapitre 5, « De l'algorithme paramétré à l'écriture d'une fonction », et 7, « Les classes et les objets »).

Copyright © 2016 Eyrolles.

Pour résoudre le niveau 2, nous allons apprendre les tâches suivantes :

- gérer la mémoire de l'ordinateur (voir les chapitres 9, « Collectionner un nombre fixe d'objets », et 10, « Collectionner un nombre indéterminé d'objets ») ;
- sauvegarder des informations pour que celles-ci ne disparaissent pas une fois l'ordinateur éteint (voir le chapitre 10, « Collectionner un nombre indéterminé d'objets »).

Le niveau 3 va nous initier aux opérations suivantes :

- calculer des statistiques (voir les chapitres 1, « Stocker une information » et 9, « Collectionner un nombre fixe d'objets ») ;
- dessiner, en particulier des histogrammes (voir le chapitre 11, « Dessiner des objets »).

Le niveau 4 va nous apprendre à manipuler des objets tels que :

- les fichiers textes ou objets, les listes (voir le chapitre 10, « Collectionner un nombre indéterminé d'objets ») ;
- les composants graphiques et les gestionnaires d'événements (voir le chapitre 12, « Créer une interface graphique »).

L'étude, étape par étape, de l'ensemble de cet ouvrage va nous permettre de réaliser cette application.

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

Résumé

En informatique, résoudre un problème c'est trouver la suite logique de tous les ordres nécessaires à la solution dudit problème. Cette suite logique est appelée **algorithme**.

La construction d'un algorithme passe par l'analyse du problème, avec pour objectif de le découper en une succession de tâches simplifiées et distinctes. Ainsi, à partir de l'énoncé clair, précis et écrit en français d'un problème, nous devons accomplir les deux opérations suivantes :

- Décomposer l'énoncé en étapes distinctes qui conduisent à l'algorithme.
- Définir les objets manipulés par l'algorithme.

Une fois l'algorithme construit, il faut « écrire le programme », c'est-à-dire **traduire** l'algorithme de façon qu'il soit compris par l'ordinateur. En effet, un programme est un algorithme traduit dans un langage compréhensible par les ordinateurs.

Un ordinateur est composé des deux éléments principaux suivants :

- La **mémoire centrale**, qui sert à mémoriser des ordres ainsi que des informations manipulées par le programme. Schématiquement, on peut dire qu'elle est composée d'emplacements repérés chacun par un nom (côté programmeur) et par une adresse (côté ordinateur).
- L'**unité centrale**, qui exécute une à une les instructions du programme dans leur ordre de lecture. Elle constitue la partie active de l'ordinateur. Ces actions, en nombre limité, sont les suivantes :
  - déposer une information dans une case mémoire ;
  - exécuter des opérations simples, telles que l'addition, la soustraction, etc. ;
  - comparer des valeurs ;
  - communiquer une information élémentaire par l'intermédiaire du clavier ou de l'écran ;
  - coder l'information.

Du fait de la technologie, toutes les informations manipulées par un ordinateur sont codées en binaire (0 ou 1). Pour s'affranchir du langage machine binaire, on fait appel à un langage de programmation dit évolué, tel que les langages Pascal, C ou Java. Un tel programme se compose d'instructions définies par le langage, dont l'enchaînement réalise la solution du problème posé.

Pour traduire ce programme dans le langage binaire directement exécutable par l'ordinateur, nous devons utiliser un programme approprié, appelé compilateur ou interpréteur. Dans cet ouvrage, nous nous proposons d'étudier comment construire un programme en prenant comme support de langage, le **langage** et le **compilateur Java**.

Copyright © 2016 Eyrolles.

Exercices

Apprendre à décomposer une tâche en sous-tâches distinctes

Exercice I.1 Écrivez la marche à suivre qui explique comment accrocher un tableau au centre d'un mur. Pour cela, vous devez :

a. Définir les objets nécessaires à la résolution du problème.

b. Établir la liste des opérations.

c. Ordonner cette liste.

Remarque Plusieurs solutions sont possibles, mais chacune doit rester logique à l'égard des hypothèses prises en a. Par exemple, un clou et une perceuse ne vont pas ensemble.

Observer et comprendre la structure d'un programme Java

Exercice I.2 Observez le programme suivant :

```
import java.util.*;
public class Premier
{
 public static void main(String [] argument)
 {
  double a ;
  Scanner lectureClavier = new Scanner(System.in);
  System.out.print("Entrer une valeur : ") ;
  a = lectureClavier.nextDouble();
  System.out.print(" Vous avez entre : " + a) ;
 }
}
```

a. Repérez les instructions définissant la fonction `main()` et celles délimitant la classe `Premier`.

b. Recherchez les instructions d'affichage.

c. Quel est le rôle de l'instruction `double a ;` ?

d. Décrivez l'exécution de ce programme, en supposant que l'utilisateur entre au clavier la valeur `10`.

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

Exercice 1.3 En suivant la structure ci-dessous et en vous aidant du programme donné à la section « Calcul de la circonférence d'un cercle », écrivez un programme qui calcule le périmètre d'un carré (rappel : périmètre = 4 × côté) :

```
public class ..... // Donner un nom à la classe
{
 public static void main(String [] argument)
 {
  // Déclaration des variables représentant le périmètre et le côté
  .....
  // Déclaration de la variable représentant la lecture au clavier
  .....
  // Afficher le message "Valeur du côté : " à l'écran
  .....
  // Lire au clavier une valeur
  // Placer cette valeur dans la variable correspondante
  .....
  // Calculer le périmètre du carré
  .....
  // Afficher le résultat
  .....
 }
}
```

Écrire un premier programme Java

Exercice 1.4 En suivant la structure de l'exercice précédent, écrivez un programme qui calcule la surface d'un rectangle (rappel : surface = largeur × longueur).

En observant la formule :

a. Combien de variables faut-il déclarer pour exécuter le calcul ?

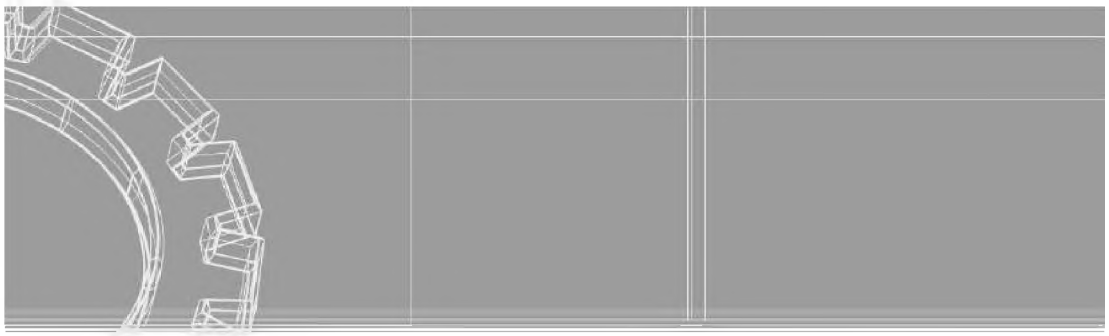
b. Combien de valeurs faut-il saisir au clavier ?

Copyright © 2016 Eyrolles.

Copyright © 2016 Eyrolles.

Partie I

Outils et techniques de base

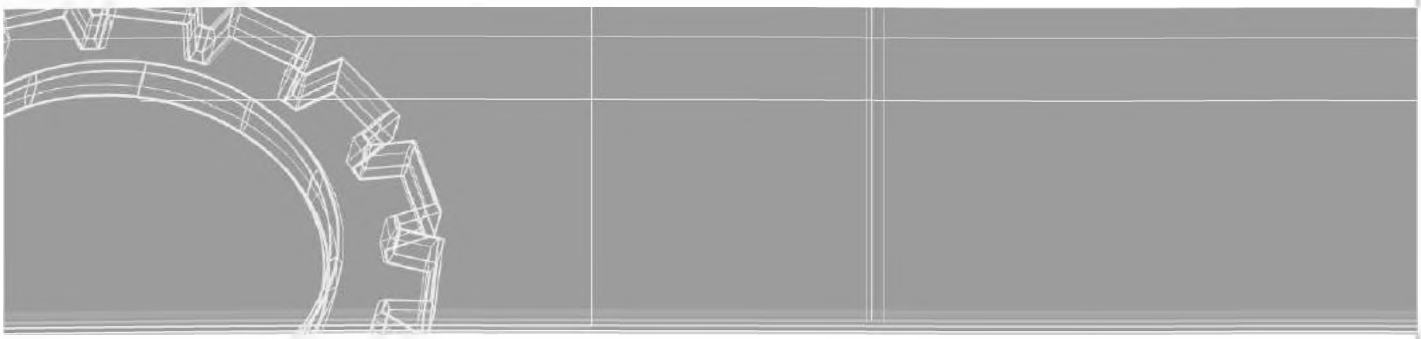


Copyright © 2016 Eyrolles.

Copyright © 2016 Eyrolles.

Chapitre 1

Stocker une information



En décrivant, au chapitre introductif, « Naissance d'un programme », l'algorithme de confection d'un café chaud non sucré, nous avons constaté que la toute première étape pour élaborer une marche à suivre consistait à déterminer les objets utiles à la résolution du problème. En effet, pour faire du café, nous devons **prendre** le café, l'eau, le filtre, etc.

De la même façon, lorsqu'un développeur d'applications conçoit un programme, il doit non pas « prendre », au sens littéral du mot, les données numériques mais **définir** ces données ainsi que les objets nécessaires à la réalisation du programme. Cette définition consiste à nommer ces objets et à décrire leur contenu afin qu'ils puissent être stockés en mémoire.

C'est pourquoi nous étudions dans ce chapitre ce qu'est une variable et comment la définir (voir section « La notion de variable »). Nous examinons ensuite, à la section « L'instruction d'affectation », comment placer une valeur dans une variable, par l'intermédiaire de l'instruction d'affectation. Enfin, nous analysons l'incidence du type des variables sur le résultat d'un calcul arithmétique (voir section « Les opérateurs arithmétiques »).

Afin de clarifier les explications, vous trouverez tout au long du chapitre des exemples simples et concis. Ces exemples ne sont pas des programmes complets mais de simples extraits, qui éclairent un point précis du concept abordé. Vous trouverez en fin de chapitre (voir section « Calculer des statistiques sur des opérations bancaires »), un programme entier qui aborde et résume toutes les notions rencontrées au fil de ce chapitre.

Copyright © 2016 Eyrolles.

La notion de variable

Une variable permet la manipulation de données et de valeurs. Elle est caractérisée par les éléments suivants :

- Un **nom**, qui sert à repérer un emplacement en mémoire dans lequel une valeur est placée. Le choix du nom d'une variable est libre. Il existe cependant des contraintes, que nous présentons à la section « Les noms de variables ».
- Un **type**, qui détermine la façon dont est traduite la valeur en code binaire ainsi que la taille de l'emplacement mémoire. Nous examinons ce concept à la section « La notion de type ». Plusieurs types simples sont prédéfinis dans le langage Java, et nous en détaillons les caractéristiques à la section « Les types de base en Java ».

Les noms de variables

Le choix des noms de variables n'est pas limité. Il est toutefois recommandé d'utiliser des noms évocateurs. Par exemple, les noms des variables utilisées dans une application qui gère les codes-barres de produits vendus en magasin sont plus certainement `article`, `prix`, `codebarre` que `xyz1`, `xyz2`, `xyz3`. Les premiers, en effet, évoquent mieux l'information stockée que les seconds.

Les contraintes suivantes sont à respecter dans l'écriture des noms de variables :

- Le premier caractère d'une variable doit obligatoirement être différent d'un chiffre.
- Aucun espace ne peut figurer dans un nom.
- Les majuscules sont différentes des minuscules, et tout nom de variable possédant une majuscule est différent du même nom écrit en minuscule.
- Les caractères `&`, `~`, `"`, `#`, `'`, `{`, `}`, `(`, `)`, `[`, `]`, `-`, `|`, `` ` ``, `\`, `^`, `@`, `=`, `%`, `*`, `?`, `:`, `/`, `§`, `!`, `<`, `>`, ainsi que `;` et `,` ne peuvent être utilisés dans l'écriture d'un nom de variable.

Tout autre caractère peut être utilisé, y compris les caractères accentués, le caractère de soulignement (_), les caractères grecs (µ) et les symboles monétaires $, _, £, etc.).

Le nombre de lettres composant le nom d'une variable est indéfini. Néanmoins, l'objectif d'un nom de variable étant de renseigner le programmeur sur le contenu de la variable, il n'est pas courant de rencontrer des noms de variables de plus de trente lettres.

Copyright © 2016 Eyrolles.

Question Parmi les variables suivantes quelles sont celles dont le nom n'est pas autorisé, et pourquoi ?

`Compte, pourquoi#pas, Num_2, -plus, Undeux, 2001espace, @adresse, VALEUR_temporaire, ah!ah!, Val$solde`

Réponse Les noms des variables suivantes ne sont pas autorisés :

`pourquoi#pas, -plus, @adresse, ah!ah!`, car les caractères `#`, `-`, `@` et `!` sont interdits.

`2001espace` car, il n'est pas possible de placer un chiffre en début de variable.

Par contre les noms de variable autorisés sont :

`Compte`, `Num_2` (" _ " et non pas " – "), `Undeux` (et non pas `un deux`), `VALEUR_temporaire`, `Val$solde`.

La notion de type

Un programme doit gérer des informations de nature diverse. Ainsi, les valeurs telles que `123` ou `2.4` sont de type numérique tandis que `Spinoza` est un mot composé de caractères. Si l'être humain sait, d'un simple coup d'œil, faire la distinction entre un nombre et un mot, l'ordinateur n'en est pas capable. Le programmeur doit donc « expliquer » à l'ordinateur la nature de chaque donnée. Cette explication passe par la notion de type.

Le type d'une valeur permet de différencier la nature de l'information stockée dans une variable. À chaque type sont associés les éléments suivants :

- Un code spécifique permettant la traduction de l'information en binaire et réciproquement.
- Un ensemble d'opérations réalisables en fonction du type de variable utilisé. Par exemple, si la division est une opération cohérente pour deux valeurs numériques, elle ne l'est pas pour deux valeurs de type caractère.
- Un intervalle de valeurs possibles dépendant du codage utilisé. Par définition, à chaque type correspond un même nombre d'octets et, par conséquent, un nombre limité de valeurs différentes.

En effet, un octet est un regroupement de 8 bits, sachant qu'un bit ne peut être qu'un `0` ou un `1`. Lorsqu'une donnée est codée sur 1 octet, elle peut prendre les valeurs `00000000` (8 zéros), ou encore `11111111` (8 un) et toutes les valeurs intermédiaires entre ces deux extrêmes (par exemple `10101010`, `11110000` ou `10010110`).

En fait, une donnée codée sur 8 bits peut, par le jeu des combinaisons de `0` et de `1`, prendre 2^8 valeurs différentes, soit 256 valeurs possibles comprises entre –128 et 127. L'intervalle [– 128, 127] est en effet composé de 256 valeurs et possède autant de valeurs positives que négatives.

Pour représenter la valeur numérique 120, un codage sur 1 octet suffit, mais pour représenter la valeur 250, 1 octet ne suffit pas, et il est nécessaire d'utiliser un codage sur 2 octets.

Copyright © 2016 Eyrolles.

Question Quel est l'intervalle de valeurs possibles pour une donnée codée sur 2 octets ?

Réponse 2 octets correspondent à 2 x 8 bits, soit 16 bits. Une donnée codée sur 16 bits peut donc prendre 2^{16} valeurs différentes, soit 65536 valeurs comprises entre –32768 et 32767.

Les types de base en Java

Chaque langage de programmation propose un ensemble de types de base permettant la manipulation de valeurs numériques entières, réelles, ou de caractères. Ces types sont :

- représentés par un mot-clé prédéfini par le langage ;
- dits **simples,** car, à un instant donné, une variable de type simple ne peut contenir qu'une et une seule valeur.

À l'opposé, il existe des types appelés types structurés, qui permettent le stockage, sous un même nom de variable, de plusieurs valeurs de même type ou non. Il s'agit des tableaux, des classes, des listes ou encore des dictionnaires. Ces types structurés sont en général définis par le programmeur. Nous les étudions en détail dans la troisième partie de cet ouvrage, intitulée « Outils et techniques orientés objet ».

Pour sélectionner un type plutôt qu'un autre, le langage Java définit huit types simples, qui appartiennent, selon ce qu'ils représentent, à l'une ou l'autre des quatre catégories suivantes : logique, caractère, entier, réel.

Catégorie logique

Il s'agit du type `boolean`. Les valeurs logiques ont deux états : `"true"` (vrai) ou `"false"` (faux). Elles ne peuvent prendre aucune autre valeur que ces deux états.

Catégorie caractère

Cette catégorie ne comprend qu'un type de base, le type `char`, qui permet de représenter les caractères isolés.

Remarque Pour décrire une suite de caractères (mots, phrases), on utilise le type `String`, qui n'est pas un type simple, mais un type structuré. Pour comprendre les types structurés tels que le type `String`, reportez-vous au chapitre 7, « Les classes et les objets », section « La classe `String`, une approche vers la notion d'objet ».

Comme nous l'avons observé dans le chapitre introductif de ce livre, toute information, toute donnée traitée par l'ordinateur n'est en réalité qu'une suite de 0 et de 1. Les caractères n'échappent pas à cette règle. Ainsi, pour décrire une variable de type `char`, l'ordinateur doit utiliser une table de correspondance qui associe à un caractère donné (voir la section « Le code Unicode » ci-après), une valeur numérique (voir la section « Codage d'un caractère »).

Copyright © 2016 Eyrolles.

Le code Unicode

L'ensemble des caractères provenant des différents alphabets internationaux (européens, africains, asiatiques, etc.) est répertorié dans une charte internationale appelée jeu de caractères Unicode. La table Unicode dresse la liste de tous les caractères utilisés dans le monde et définit pour chacun d'entre eux une valeur numérique unique appelée *code-point*.

Ainsi, par exemple, dans la table Unicode, le caractère K majuscule a pour code-point 004B et le caractère k minuscule a pour code-point 006B. Plus précisément, les 128 premières valeurs de la table Unicode sont organisées comme suit :

Caractère	NUL	DLE	SP	0	@	P	`	p
Unicode	0000	0010	0020	0030	0040	0050	0060	0070
Caractère	SOH	DC1	!	1	A	Q	a	q
Unicode	0001	0011	0021	0031	0041	0051	0061	0071
Caractère	STX	DC2	"	2	B	R	b	r
Unicode	0002	0012	0022	0032	0042	0052	0062	0072
Caractère	ETX	DC3	#	3	C	S	c	s
Unicode	0003	0013	0023	0033	0043	0053	0063	0073
Caractère	EOT	DC4	$	4	D	T	d	t
Unicode	0004	0014	0024	0034	0044	0054	0064	0074
Caractère	ENQ	NAK	%	5	E	U	e	u
Unicode	0005	0015	0025	0035	0045	0055	0065	0075
Caractère	ACK	SYN	&	6	F	V	f	v
Unicode	0006	0016	0026	0036	0046	0056	0066	0076
Caractère	BEL	ETB	'	7	G	W	g	w
Unicode	0007	0017	0027	0037	0047	0057	0067	0077
Caractère	BS	CAN	(	8	H	X	h	x
Unicode	0008	0018	0028	0038	0048	0058	0068	0078
Caractère	HT	EM	)	9	I	Y	i	y
Unicode	0009	0019	0029	0039	0049	0059	0069	0079
Caractère	LF	SUB	*	:	J	Z	j	z
Unicode	000A	001A	002A	003A	004A	005A	006A	007A
Caractère	VT	ESC	+	;	K	[	k	{
Unicode	000B	001B	002B	003B	004B	005B	006B	007B
Caractère	FF	FS	,	<	L	\	l	|
Unicode	000C	001C	002C	003C	004C	005C	006C	007C
Caractère	CR	GS	-	=	M	]	m	}
Unicode	000D	001D	002D	003D	004D	005D	006D	007D
Caractère	SO	RS	.	>	N	^	n	~
Unicode	000E	001E	002E	003E	004E	005E	006E	007E
Caractère	SI	US	/	?	O	_	o	DEL
Unicode	000F	001F	002f	003F	004F	005F	006F	007F

Copyright © 2016 Eyrolles.

- Les valeurs comprises entre `0000` et `001F` correspondent à des caractères qui ne peuvent être affichés, comme le caractère de tabulation `HT`, le saut de ligne `CR`, ou le bip sonore `BEL`.
- Les valeurs comprises entre `0020` et `007F` correspondent aux caractères du code ASCII (*American Standard Code for Information Interchange*) qui était, avant la mise en place de l'Unicode, le code définissant tout caractère. Dans cet intervalle, tous les caractères de base sont définis, c'est-à-dire l'ensemble des lettres de l'alphabet, en minuscules et en majuscules, ainsi que les signes de ponctuation et les symboles mathématiques.
- Les valeurs suivantes (`0080` à `FFFF`) correspondent à des caractères spéciaux tels que les caractères accentués, les caractères de l'alphabet russe, les idéogrammes chinois, les symboles du copyright et de l'euro, etc.

Extension Web Pour connaître le code Unicode d'un caractère de l'alphabet latin, consultez les fichiers `Unicode0000a007F.pdf` et `Unicode0080a00FF.pdf` placés sur l'extension Web de l'ouvrage. Pour connaître le code Unicode d'un caractère chinois ou cyrillique, rendez-vous sur le site *www.unicode.org*.

Née en 1991, la table Unicode est aujourd'hui le standard des jeux de caractères. Il existe cependant d'autres jeux de caractères comme l'ISO-8859-1, ISO-8859-15, ou encore Windows-1252 qui restent encore très utilisés, notamment par les systèmes d'exploitation des ordinateurs.

Codage d'un caractère

Quel que soit le jeu de caractères choisi (alphabet latin, alphabet russe, idéogrammes chinois, signes cunéiformes…), chaque caractère est représenté dans la mémoire de l'ordinateur par une valeur numérique unique : cette opération s'appelle le codage de caractères ou encore l'encodage.

Selon la taille du jeu de caractères utilisé, l'encodage pourra s'effectuer sur 1, 2 ou 4 octets. La valeur numérique attribuée à un caractère est alors calculée à l'aide de son code-point, défini dans la table Unicode, et codée sur 1, 2 ou 4 octets.

Ainsi, lorsque l'encodage s'effectue sur :

- 1 octet, on dispose au total de 256 valeurs. Cette forme d'encodage suffit par exemple pour coder l'ensemble de l'alphabet latin. Le caractère `k` a alors pour code `6B`.
- 2 octets, le nombre de valeurs disponibles est plus important (62 536 valeurs), ce qui permet de coder par exemple la totalité des idéogrammes chinois. Dans ce cas, le caractère `k` aura pour code `00 6B`.
- 4 octets, le nombre de valeurs disponibles s'élève à 2^{32}. Le caractère `k` a alors pour code `00 00 00 6B`.

Suivant le nombre d'octets utilisés (1, 2 ou 4), ces formes d'encodage se nomment respectivement UTF-8 (Unicode Transformation Format), UTF-16 et UTF-32.

Le langage Java utilise par défaut la forme d'encodage sur deux octets (UTF-16), ce qui couvre la plupart des systèmes d'écriture utilisés dans le monde.

Copyright © 2016 Eyrolles.

Remarque Avec la forme d'encodage UTF-16, le code d'un caractère s'écrit en Java `\u` suivi du code-point codé sur 2 octets. Ainsi, le code Unicode de la lettre k minuscule s'écrit `\u006B` et celui de la lettre K, `\u004B`.

Catégorie entier

Cette catégorie contient quatre types distincts : `byte`, `short`, `int`, `long`. Chacun de ces types autorise la manipulation de valeurs numériques entières, positives ou négatives. Leur différence réside essentiellement dans le nombre d'octets utilisés pour coder le contenu de la variable.

Type	Nombre d'octets	Éventail de valeurs
`byte`	1 octet	De - 128 à 127
`short`	2 octets	De - 32 768 à 32 767
`int`	4 octets	De - 2 147 483 648 à 2 147 483 647
`long`	8 octets	De - 9 223 372 036 854 775 808 à 9 223 372 036 854 775 807

Dans certains cas, il est intéressant de représenter une valeur entière sous forme octale ou hexadécimale comme pour l'affichage des caractères de la table Unicode.

Pour en savoir plus Pour calculer une valeur dans le système hexadécimal, reportez-vous au chapitre 2, « Communiquer une information », section « Afficher les caractères accentués ».

Valeur décimale	Valeur octale	Valeur hexadécimale
45	055	0x2d

Pour représenter un nombre sous forme octale, il est nécessaire de placer un zéro au début du nombre. Pour la représentation sous forme hexadécimale, les caractères `0x` doivent être placés en début de valeur.

Remarque Dans le langage Java, tous les types de la catégorie entier ont un signe (+ ou -).

Catégorie réel (flottant)

La catégorie réel permet l'emploi de nombres à virgule, appelés nombres réels ou encore flottants.

Deux types composent cette catégorie, le type `float` et le type `double`. Une expression numérique de cette catégorie peut s'écrire en notation décimale ou exponentielle.

- La notation décimale contient obligatoirement un point symbolisant le caractère « virgule » du chiffre à virgule. Les valeurs `67.3`, `-3.` ou `.64` sont des valeurs réelles utilisant la notation décimale.

Copyright © 2016 Eyrolles.

- La notation exponentielle utilise la lettre `E` pour déterminer où se trouve la valeur de l'exposant (puissance de 10). Les valeurs 8.76E4 et 6.5E–12 sont des valeurs utilisant la notation exponentielle.

Dans les deux cas, le nombre réel est suivi de la lettre `F` (pour `float`) ou `D` (pour `double`). Les caractères minuscules `f` ou `d` sont également autorisés. La distinction entre `float` et `double` s'effectue sur le nombre d'octets utilisés pour coder l'information. Il en résulte une précision plus ou moins grande suivant le type employé.

Type	Nombre d'octets	Éventail des valeurs
`float`	4 octets	de `1.40239846e-45F` à `3.402823347e38F`
`double`	8 octets	de `4.94065645841246544e-324D` à `1.79769313486231570e308D`

En langage Java, toute valeur numérique réelle est définie par défaut en double précision. Par conséquent, la lettre `d` (ou `D`) placée en fin de valeur n'est pas nécessaire. Par contre, dès qu'on utilise une variable `float`, la lettre `f` (ou `F`) est indispensable, sous peine d'erreur de compilation.

Question

Avec quel type de variables les valeurs suivantes peuvent-elles être définies ?

1. `2.15F` et `6.76f`
2. `1.35E22` et `463.4E+234D`

Réponse

1. Les valeurs `2.15F` et `6.76f` sont des nombres à virgule. La lettre F ou f indique à l'interpréteur Java que ces valeurs sont de simple précision. Elles peuvent donc être stockées dans des variables de type `float`.
2. Les valeurs `1.35E22` et `463.4E+234D` sont des nombres à virgule de double précision. La lettre F étant absente, les valeurs sont considérées par l'interpréteur comme étant de type `double`.

Comment choisir un type de variable plutôt qu'un autre ?

Sachant qu'une variable de type `int` (codée sur 4 octets) peut prendre toutes les valeurs de l'intervalle [–2147483648, 2147483647] et donc prendre, en particulier, toutes les valeurs comprises entre –32768 et 32767 (type `short`) ou même entre –128 et 127 (type `byte`), posons-nous les questions suivantes :

- Pourquoi ne pas déclarer toutes les variables entières d'un programme en type `long` (le type `long` nous offrant le plus grand choix de valeurs entières) ?
- Pourquoi ne pas déclarer les variables réelles d'un programme en type `double` plutôt qu'en `float` ?

Copyright © 2016 Eyrolles.

Pour répondre à ces questions, examinons le nombre d'octets utilisés par un programme de gestion de comptes bancaires. Pour simplifier, supposons que le programme garde en mémoire les 10 dernières opérations bancaires et le solde de chaque compte. Imaginons enfin que notre banque gère 50 000 comptes.

Pour stocker les 10 dernières opérations, nous devons déclarer 10 variables plus 1 pour le solde du compte, soit 11 variables. Les valeurs sont des montants en euros et cents, donc des valeurs réelles.

- Si nous déclarons l'ensemble de ces variables en type `double` (8 octets), le programme utilise alors 50 000 × 11 × 8 octets, soit 4 400 000 octets, soit 4,4 mégaoctets de la mémoire de l'ordinateur.
- Si nous choisissons de prendre des variables de type `float` (ce qui reste cohérent, puisque les montants en euros n'ont pas besoin d'être d'une précision extrême), notre programme n'utilise plus que 2,2 mégaoctets, soit deux fois moins que précédemment.

Bien entendu, cet exemple simpliste n'a pour seul objectif que de montrer l'effet du choix du type de variable sur le taux d'occupation de la mémoire de l'ordinateur. Il existe, en réalité, un grand nombre de techniques pour optimiser la gestion de la mémoire de l'ordinateur.

Remarque La première démarche pour gérer au mieux la mémoire de l'ordinateur consiste à bien choisir le type de ses variables. Si l'on sait que, par définition, une variable ne dépasse jamais, pour un programme donné, la valeur numérique 120, celle-ci doit être déclarée avec le type `byte`.

Question À quel type peut-on associer une variable représentant l'âge d'une personne ?

Réponse Le type `byte` suffit, puisque les valeurs d'un `byte` varient entre `-128` et `127`. Raisonnablement, on peut dire que 127 ans reste encore aujourd'hui un âge limite pour l'espèce humaine.

Question À quel type peut-on associer une variable représentant le nombre correspondant aux années ?

Réponse Tout dépend du calendrier utilisé. Pour le calendrier chrétien, le type `short` suffit, puisque les valeurs d'un `short` varient entre `-32768` et `32767`. La question reste à savoir s'il convient de penser au futur bug de l'an 32767.

Déclarer une variable

La définition d'une variable dans un programme est réalisée par l'intermédiaire de l'instruction de déclaration des variables. Au cours de cette instruction, le programmeur donne le type et le nom de la variable. Pour déclarer une variable, il suffit d'écrire l'instruction selon la syntaxe suivante :

```
type nomdevariable ;
```

Copyright © 2016 Eyrolles.

ou

```
type nomdevariable1, nomdevariable2 ;
```

où `type` correspond à l'un des mots-clés à choisir parmi ceux donnés aux sections précédentes (`boolean`, `char`, `String`, `byte`, `short`, `int`, `long`, `float` ou `double`). Si deux variables de même type sont à déclarer, il n'est pas besoin de répéter le type, une virgule séparant les deux noms suffisant à les distinguer.

Pour expliquer à l'ordinateur que l'instruction de déclaration est terminée pour le type donné, un point virgule (`;`) est placé obligatoirement à la fin de la ligne d'instruction.

Question Quel est le rôle des instructions suivantes ?

```
float   f1, f2 ;
long    CodeBar ;
int     test ;
char     choix, tmp ;
boolean  OK ;
```

Réponse L'instruction :

```
float   f1, f2 ;
```

fait que deux variables de type `float` sont déclarées.

```
long    CodeBar ;
```

permet de déclarer une variable de type `long`.

```
int     test ;
```

permet de déclarer une variable de type `int`.

```
char     choix, tmp ;
```

fait que deux variables de type `char` sont déclarées.

```
boolean  OK ;
```

permet de déclarer une variable de type `boolean`.

Remarque La virgule permet de séparer les deux noms des variables lorsque celles-ci sont de même type.

Les instructions de déclaration peuvent être placées indifféremment au début ou en cours de programme. Une fois la variable déclarée, l'interpréteur Java réserve, au cours de l'exécution du programme, un emplacement mémoire correspondant en taille à celle demandée par le

Copyright © 2016 Eyrolles.

type. Il associe ensuite le nom de la variable à l'adresse de l'emplacement mémoire. À cette étape du programme, observons que l'emplacement ainsi défini est vide.

Observons cependant que si l'on souhaite afficher le contenu d'une variable sans y avoir préalablement déposé de valeur, le compilateur émet le message d'erreur suivant : `Variable may not have been initialized`. Cette erreur indique que la variable dont on souhaite afficher le contenu n'a pas été initialisée. Comme l'interpréteur Java ne peut afficher un emplacement mémoire vide, l'exécution du programme n'est pas possible.

Pour en savoir plus Nous verrons à partir du chapitre 6, « Fonctions, notions avancées » et au chapitre 7, « Les classes et les objets » que la position de déclaration des variables influence fortement le comportement d'une application.

L'instruction d'affectation

Une fois la variable déclarée, il est nécessaire de stocker une valeur à l'emplacement mémoire désigné. Pour ce faire, nous utilisons l'instruction d'affectation, qui nous permet d'initialiser ou de modifier, en cours d'exécution du programme, le contenu de l'emplacement mémoire (le contenu d'une variable n'étant, par définition, pas constant).

Rôle et mécanisme de l'affectation

L'affectation est le mécanisme qui permet de placer une valeur dans un emplacement mémoire. Elle a pour forme :

```
Variable = Valeur ;
```

ou encore,

```
Variable = Expression mathématique ;
```

Le signe égal (`=`) symbolise le fait qu'une valeur est placée dans une variable. Pour éviter toute confusion sur ce signe mathématique bien connu, nous prendrons l'habitude de le traduire par les termes `prend la valeur`.

Examinons les exemples suivants, en supposant que les variables `n` et `p` soient déclarées de type entier :

```
n = 4 ;      // n prend la valeur 4
p = 5*n+1 ; // calcule la valeur de l'expression mathématique soit
             // 5*4+1 et range la valeur obtenue dans la variable
             // représentée par p.
```

Copyright © 2016 Eyrolles.

L'instruction d'affectation s'effectue dans l'ordre suivant :

1. calcule la valeur de l'expression figurant à droite du signe égal ;
2. range le résultat obtenu dans la variable mentionnée à gauche du signe égal.

Remarque

La variable placée à droite du signe égal (=) n'est jamais modifiée, alors que celle qui est à gauche l'est toujours. Comme une variable de type simple ne peut stocker qu'une seule valeur à la fois, si la variable située à gauche possède une valeur avant l'affectation, cette valeur est purement et simplement remplacée par la valeur située à droite du signe `égal (=)`.

Exemple

```
a = 1 ;
b = a + 3 ;
a = 3 ;
```

Lorsqu'on débute en programmation, une bonne méthode pour comprendre ce que réalise un programme consiste à écrire, pour chaque instruction exécutée, un état de toutes les variables déclarées. Il suffit pour cela de construire un tableau dont chaque colonne représente une variable déclarée dans le programme et chaque ligne une instruction de ce même programme. Soit, pour notre exemple :

Instruction	a	b
`a = 1`	1	-
`b = a + 3`	1	4
`a = 3`	3	4

Le tableau est composé des deux colonnes `a` et `b` et des trois lignes associées aux instructions d'affectation du programme. Ce tableau montre que les instructions `a = 1` et `a = 3` font que la valeur initiale de `a (1)` est effacée et écrasée par la valeur `3`.

Déclaration et affectation

Comme nous l'avons vu à la section « Déclarer une variable », la déclaration est utilisée pour réserver un emplacement mémoire. Une fois réservé, l'emplacement reste vide jusqu'à ce qu'une valeur y soit placée par l'intermédiaire de l'affectation.

Il est cependant risqué de déclarer une variable sans lui donner de valeur initiale. En effet, le compilateur Java vérifie strictement si toutes les variables contiennent une valeur ou non. Une erreur de compilation est détectée dès qu'une variable à afficher ne contient pas de valeur à un moment donné du programme.

Copyright © 2016 Eyrolles.

Question Quelles sont les valeurs des variables `prix`, `tva` et `total` après exécution des instructions suivantes :

```
prix = 20 ;
tva = 18.6 ;
total = prix + prix*tva / 100;
```

Réponse

Instruction	prix	tva	total
`prix = 20 ;`	20	-	-
`tva = 18.6 ;`	20	18.6	-
`total = prix + prix * tva / 100 ;`	20	18.6	23.72 (20 + 20 * 18.6 / 100)

Initialiser une variable

Pour éviter toute erreur de compilation, une bonne habitude consiste à initialiser toutes les variables au moment de leur déclaration, en procédant de la façon suivante :

```
float    f1 = 0.0f, f2 = 1.2f ;  // Initialisation de deux float
long     CodeBar = 123456789 ;   // Initialisation d'un long
int      test = 0 ;    // Initialisation d'une variable de type int
boolean  OK = true ;   // Initialisation d'un boolean
```

De cette façon, les variables `f1`, `f2`, `CodeBar` et `OK` sont déclarées. Le compilateur réserve un emplacement mémoire pour chacune d'entre elles. Grâce au signe d'affectation, le compilateur place dans chacun des emplacements mémoire respectifs les valeurs données.

Initialiser une variable de type char

Les variables de type `char` s'initialisent d'une façon particulière. Supposons que l'on souhaite déclarer et placer le caractère `n` dans une variable `choix` de type `char`. Pour cela, écrivons l'instruction de déclaration et d'initialisation suivante :

```
char choix = n ;
```

Pour le compilateur, cette instruction est problématique, car il considère `n` non pas comme le « caractère `n` » mais comme une variable appelée `n`.

Pour lever cette ambiguïté, nous devons entourer le caractère `n` d'apostrophes, de la façon suivante :

```
char choix = 'n' ;
```

Ainsi, des données telles que `'a'`, `'*'`, `'$'`, `'3'`, `':'` ou `'?'` sont considérées comme des caractères.

Copyright © 2016 Eyrolles.

Par contre `c = 'ab'` ne peut s'écrire, car `'ab'` n'est pas un caractère mais un mot de deux caractères. Nous devons, dans ce cas, utiliser une variable de type `String`.

Pour en savoir plus Le type `String` est décrit au chapitre 7, « Les classes et les objets », dans la section « La classe String, une approche vers la notion d'objet ».

Question Quelles sont les valeurs des variables `c1`, `c2`, `c3` et `OK` après exécution des instructions suivantes :

```
char c1 = 'o', c2 = 'u', c3 = 'i' ;
String OK ;
OK = c1 + c2 + c3 ;
```

Réponse

Instruction	c1	c2	c3	OK
`char c1 = 'o' ;`	'o'	-	-	-
`char c2 = 'u' ;`	'o'	'u'	-	-
`char c3 = 'i' ;`	'o'	'u'	'i'	-
`OK = c1 + c2 + c3 ;`	'o'	'u'	'i'	"oui"

Remarque La variable `OK` est de type `String` puisqu'elle est composée de plusieurs caractères.

Quelques confusions à éviter

Le symbole de l'affectation est le signe égal (=). Ce signe, très largement utilisé dans l'écriture d'équations mathématiques, est source de confusion lorsqu'il est employé à contre-sens.

Pour mieux nous faire comprendre, étudions trois cas :

```
1. a = a + 1 ;
```

Si cette expression est impossible à écrire d'un point de vue mathématique, elle est très largement utilisée dans le langage informatique. Elle signifie :

- calculer l'expression `a + 1` ;
- ranger le résultat dans `a`.

Ce qui revient à augmenter de `1` la valeur de `a`.

```
2. a + 5 = 3 ;
```

Copyright © 2016 Eyrolles.

Cette expression n'a aucun sens d'un point de vue informatique. Il n'est pas possible de placer une valeur à l'intérieur d'une expression mathématique, puisque aucun emplacement mémoire n'est attribué à une expression mathématique.

```
3. a = b ; et b = a ;
```

À l'inverse de l'écriture mathématique, ces deux instructions ne sont pas équivalentes. La première place le contenu de b dans a, tandis que la seconde place le contenu de a dans b.

Échanger les valeurs de deux variables

Nous souhaitons échanger les valeurs de deux variables de même type, appelées a et b ; c'est-à-dire que nous voulons que a prenne la valeur de b et que b prenne celle de a. La pratique courante de l'écriture des expressions mathématiques fait que, dans un premier temps, nous écrivions les instructions suivantes :

```
a = b ;
b = a ;
```

Vérifions sur un exemple si l'exécution de ces deux instructions échange les valeurs de a et de b. Pour cela, supposons que les variables a et b contiennent initialement respectivement 2 et 8.

	a	b
valeur initiale	2	8
a = b	8	8
b = a	8	8

Du fait du mécanisme de l'affectation, la première instruction a = b détruit la valeur de a en plaçant la valeur de b dans la case mémoire a. Lorsque la seconde instruction b = a est réalisée, la valeur placée dans la variable b est celle contenue à cet instant dans la variable a, c'est-à-dire la valeur de b. Il n'y a donc pas échange, car la valeur de a a disparu par écrasement lors de l'exécution de la première instruction.

Une solution consiste à utiliser une variable supplémentaire, destinée à contenir temporairement une copie de la valeur de a, avant que cette dernière soit écrasée par la valeur de b.

Remarque

Pour évoquer le caractère temporaire de la copie, nous appelons cette nouvelle variable tmp, nous aurions pu choisir tout aussi bien tempo, ttt ou toto.

Voici le déroulement des opérations :

```
tmp = a ;
a = b ;
b = tmp ;
```

Copyright © 2016 Eyrolles.

Vérifions qu'il y a réellement échange, en supposant que nos variables `a` et `b` contiennent initialement respectivement `2` et `8`.

	a	b	tmp
`valeur initiale`	2	8	_
`tmp = a`	2	8	2
`a = b`	8	8	2
`b = tmp`	8	2	2

À la lecture de ce tableau, nous constatons qu'il y a bien échange des valeurs entre `a` et `b`. La valeur de `a` est copiée dans un premier temps dans la variable `tmp`. La valeur de `a` peut dès lors être effacée par celle de `b`. Pour finir, grâce à la variable `tmp`, la variable `b` récupère l'ancienne valeur de `a`.

Pour en savoir plus Une autre solution vous est proposée dans la feuille d'exercices, à la section « Comprendre le mécanisme d'échange de valeurs », située à la fin du chapitre.

Les opérateurs arithmétiques

Écrire un programme ne consiste pas uniquement à échanger des valeurs, c'est aussi calculer des équations mathématiques plus ou moins complexes. Pour exprimer une opération, le langage Java utilise des caractères qui symbolisent les opérateurs arithmétiques.

Symbole	Opération
`+`	Addition
`-`	Soustraction
`*`	Multiplication
`/`	Division
`%`	Modulo

Exemple

Soient `a`, `b`, `c` trois variables de même type.

- L'opération d'addition s'écrit : `a = b + 4`.
- L'opération de soustraction s'écrit : `a = b - 5`.

Copyright © 2016 Eyrolles.

- L'opération de division s'écrit : `a = b / 2` et non pas `a =` $\frac{b}{2}$.
- L'opération de multiplication s'écrit : `a = b * 4`
- et non pas `a = 4b` ou `a = 4 x b`.
- L'opération de modulo s'écrit : `a = b % 3`.

Le modulo d'une valeur correspond au reste de la division entière. Ainsi : `5 % 2 = 1`.

Il s'agit de calculer la division en s'arrêtant dès que la valeur du reste devient inférieure au diviseur, de façon à trouver un résultat en nombre entier. L'opérateur `%` n'existe pas pour les réels, pour lesquels la notion de division entière n'existe pas.

Question Les opérations suivantes sont-elles valides ?

```
delta = b2 - 4ac ;
```

`z =` $\frac{b}{2}$ `+ 3%xa ;`

Réponse Aucune des deux opérations n'est valide.

`delta = b2 - 4ac ;` doit s'écrire `delta = b * b - 4 * a * c ;`

`z =` $\frac{b}{2}$ `+ 3%xa ;` doit s'écrire `z = b / 2 + 3 % x * a`

En supposant que *x* corresponde à une variable de type entier et non pas réel. En effet, le modulo est une opération valide uniquement pour les entiers.

L'ensemble de ces opérateurs est utilisé pour calculer des expressions mathématiques courantes. Le résultat de ces expressions n'est cependant pas toujours celui auquel on s'attend. Trois phénomènes ont une influence non négligeable sur la valeur du résultat d'un calcul. Ce sont :

- la priorité des opérateurs entre eux ;
- le type d'une expression mathématique ;
- la transformation de types.

La priorité des opérateurs entre eux

Lorsqu'une expression arithmétique est composée de plusieurs opérations, l'ordinateur doit pouvoir déterminer quel est l'ordre des opérations à effectuer. Le calcul de l'expression `a - b / c * d` peut signifier *a priori* :

- calculer la soustraction puis la division et pour finir la multiplication, soit le calcul : `((a - b) / c) * d ;`
- calculer la multiplication puis la division et pour finir la soustraction, c'est-à-dire l'expression : `a - (b / (c * d))`.

Copyright © 2016 Eyrolles.

Afin d'éviter toute ambiguïté, il existe des règles de priorité entre les opérateurs, règles basées sur la définition de deux groupes d'opérateurs.

Groupe 1	Groupe 2
+ -	* / %

Les groupes étant ainsi définis, les opérations sont réalisées sachant que :

- Dans un même groupe, l'opération se fait dans l'ordre d'apparition des opérateurs (sens de lecture).
- Le deuxième groupe a priorité sur le premier.

L'expression `a - b / c * d` est calculée de la façon suivante :

Priorité	Opérateur	
Groupe 2	/	Le groupe 2 a priorité sur le groupe 1, et la division apparaît dans le sens de la lecture avant la multiplication.
Groupe 2	*	Le groupe 2 a priorité sur le groupe 1, et la multiplication suit la division.
Groupe 1	-	La soustraction est la dernière opération à exécuter, car elle est du groupe 1.

Cela signifie que l'expression est calculée de la façon suivante :

```
a - (b / c * d)
```

Remarque Les parenthèses permettent de modifier les règles de priorité en forçant le calcul préalable de l'expression qui se trouve à l'intérieur des parenthèses. Elles offrent en outre une meilleure lisibilité de l'expression.

Le type d'une expression mathématique

Le résultat d'une expression mathématique peut être déterminé à partir du type de variables (termes) qui composent l'expression.

Terme	Opération	Terme	Résultat
Entier	+ - * / %	Entier	Entier
Réel	+ - * /	Réel	Réel

Copyright © 2016 Eyrolles.

De ce fait, pour un même calcul, le résultat diffère selon qu'il est effectué à l'aide de variables de type réel ou de type entier.

Exemple : diviser deux entiers

```
int x = 5 , y = 2, z ;
z = x / y ;
```

	x	y	z
valeur initiale	5	2	-
`z = x / y`	5	2	2

Ici, toutes les variables déclarées sont de type entier. Par conséquent, l'opération effectuée a pour résultat une valeur entière, même si l'opération demandée n'a pas forcément un résultat entier. Soit, pour notre exemple, 2 et non 2.5. Cela ne correspond pas toujours au résultat attendu par le programmeur débutant.

Question Que vaut la variable `résultat` après exécution des instructions suivantes :

```
int résultat, premier = 5, second = 3, coefficient = 2 ;
résultat = coefficient * premier / second ;
```

Réponse La multiplication et la division appartiennent au même groupe, les opérations sont donc réalisées dans le sens de la lecture. La multiplication `coefficient * premier` donne pour résultat `10`, puis la division de `10` par `second` donne `3`, puisque les deux valeurs sont entières. La variable `résultat` a donc pour valeur `3`.

Question Que vaut la variable `résultat` après exécution des instructions suivantes :

```
int résultat, premier = 5, second = 3, coefficient = 2 ;
résultat = premier / second * coefficient;
```

Réponse La multiplication et la division appartiennent au même groupe, les opérations sont donc réalisées dans le sens de la lecture. La division `premier / second` a pour résultat `1`, puisque les deux valeurs sont entières. Puis la multiplication de `1` par `coefficient` donne `2`. La variable `résultat` a donc pour valeur `2`.

Remarque À travers ces deux questions-réponses, nous observons que l'ordre des opérations et le type des données utilisées à une forte influence sur le résultat du calcul effectué.

Copyright © 2016 Eyrolles.

Exemple : diviser deux réels

```
double x = 5 , y = 2, z ;
z = x / y ;
```

	x	y	z
valeur initiale	5	2	-
`z = x / y`	5	2	2.5

Ici, toutes les variables déclarées sont de type réel. Par conséquent, l'opération effectuée donne un résultat de type réel. Ce résultat correspond à la valeur généralement attendue de ce type d'opération.

Question Que vaut la variable `résultat` après exécution des instructions suivantes :

```
double résultat, premier = 5, second = 3, coefficient = 2 ;
résultat = coefficient * premier / second ;
```

Réponse La multiplication et la division appartiennent au même groupe, les opérations sont donc réalisées dans le sens de la lecture. La multiplication `coefficient * premier` donne pour résultat `10.0`, puis la division de `10.0` par `second` donne `3.3333333`, puisque les deux valeurs sont réelles. La variable `résultat` a donc pour valeur `3.3333333`.

Question Que vaut la variable `résultat` après exécution des instructions suivantes :

```
double résultat, premier = 5, second = 3, coefficient = 2 ;
résultat = premier / second * coefficient;
```

Réponse La multiplication et la division appartiennent au même groupe, les opérations sont donc réalisées dans le sens de la lecture. La division `premier / second` a pour résultat `1.6666666`, puisque les deux valeurs sont réelles. Puis la multiplication de `1.6666666` par `coefficient` donne `3.3333333`. La variable `résultat` a donc pour valeur `3.3333333`.

La transformation de types

Les termes d'une opération ne sont pas nécessairement tous du même type. Pour écrire une opération, toutes les combinaisons entre les différentes catégories de types peuvent se présenter.

Terme	Opération	Terme	Résultat
`byte`	`+ - * /`	`int`	`int`
`int`	`+ - * /`	`double`	`double`

Copyright © 2016 Eyrolles.

L'ordinateur ne sait calculer une expression mathématique que lorsque toutes les variables de l'expression sont du même type. En effet, les opérateurs arithmétiques ne sont définis que pour des variables de type identique.

Lorsque tel n'est pas le cas, c'est-à-dire si l'expression est mixte, l'ordinateur doit transformer le type de certaines variables pour que tous les membres de l'expression deviennent de même type.

Cette transformation, appelée **conversion d'ajustement de type**, se réalise suivant une hiérarchie bien déterminée, qui permet de ne pas perdre d'information. On dit que le compilateur respecte l'intégralité des données.

La conversion d'un nombre réel en nombre entier, par exemple, ne peut se réaliser qu'en supprimant les nombres situés après la virgule et en ne gardant que la partie entière du nombre. Une telle conversion ne garantit pas l'intégralité des données car il y a perte de données.

C'est pourquoi, du fait du codage des données et du nombre d'octets utilisé pour ce codage, le compilateur effectue automatiquement la conversion des données selon l'ordre suivant :

```
byte -> short -> int -> long -> float -> double
```

De cette façon, il est toujours possible de convertir un `byte` en `long` ou un `int` en `float`. Il est également possible de convertir un `char` en `int`. Par contre, il est impossible de transformer un `float` en `short` sans perte d'information.

Exemple

```
int a = 4, result_int ;
float x = 2.0f, result_float ;
result_float = a / x ;
result_int = a / x ;
```

	a	x	result_float	result_int
`a = 4`	4	_	_	_
`x = 2.0f`	4	2.0f	_	_
`result_float = a/x`	4	2.0f	2.0f	_
`result_int = a/x`	4	2.0f	_	Impossible dès la compilation

La troisième instruction montre que le calcul d'une opération dont les termes sont de type `int` et `float` donne pour résultat un `float`. La dernière instruction révèle que, si le résultat d'une opération est de type `float`, il n'est pas possible de le stocker dans une variable de type `int`. En effet, la division d'un entier par un réel est une opération toujours possible à réaliser (le résultat est de type réel), mais l'affectation directe de ce résultat dans une variable entière est impossible du fait que la conversion entraîne une perte d'information.

Copyright © 2016 Eyrolles.

Une telle instruction provoque à la compilation une erreur dont le message est : `Incompatible type for =. Explicit cast needed to convert float to int.` Cela signifie : « Type incompatible de part et d'autre du signe =. Pour convertir un `float` en `int`, il est nécessaire de le formuler explicitement par l'intermédiaire d'un cast. »

Le cast

La conversion avec perte d'information est autorisée dans certains cas grâce au mécanisme du *cast*. Il peut être utile de transformer un nombre réel en entier, par exemple pour calculer sa partie entière. Pour cela, le compilateur demande de convertir explicitement les termes de l'opération dans le type souhaité en plaçant devant la variable ou l'opération le type de conversion désiré. Ainsi, pour transformer un `float` en `int`, il suffit de placer le terme `(int)` devant la variable ou l'opération de type `float`.

Exemple

```
int a = 5, result ;
float x = 2.0f ;
result1 = (int) a / x ;
```

	a	x	result1
`a = 5`	5	_	_
`x = 2.0f`	5	2.0f	_
`result1 = (int) a / x`	5	2.0f	2

La dernière instruction montre que la conversion `float` vers `int` est autorisée malgré la perte d'information (le chiffre 5 placé après la virgule disparaît). Cette conversion n'est possible que si elle est précisément indiquée au compilateur.

Question Que vaut la variable `résultat` après exécution des instructions suivantes :

```
int premier = 5, second = 3, coefficient = 2 ;
double résultat ;
résultat = (double) coefficient * premier / second ;
```

Réponse Le mécanisme de cast transforme la variable `coefficient` en `double`. Ensuite, la multiplication et la division appartenant au même groupe, les opérations sont réalisées dans le sens de la lecture. La multiplication `coefficient * premier` donne pour résultat `10.0` (de type `double`), puis la division de `10.0` par `second` donne `3.3333333`, puisque `10.0` est une valeur réelle. La variable `résultat` a donc pour valeur `3.3333333`.

Copyright © 2016 Eyrolles.

Question Que vaut la variable `résultat` après exécution des instructions suivantes :

```
int premier = 5, second = 3, coefficient = 2 ;
double résultat ;
résultat = (double) (coefficient * premier / second);
```

Réponse Les parenthèses entourant l'expression `(coefficient * premier / second)` font que cette expression est calculée avant d'être transformée en `double`. La multiplication et la division appartenant au même groupe, les opérations sont réalisées dans le sens de la lecture. La multiplication `coefficient * premier` donne pour résultat `10`, puis la division de `10` par `second` donne `3`, puisque les deux valeurs sont entières. Ce résultat est ensuite transformé en `double` grâce au mécanisme du `cast`. La variable `résultat` a donc pour valeur finale `3.0`.

Calculer des statistiques sur des opérations bancaires

Pour résumer en pratique l'ensemble des notions abordées dans ce chapitre, nous allons écrire un programme, dont le sujet se rapporte au thème du projet énoncé à la fin du chapitre introductif, « Naissance d'un programme ».

Cahier des charges

L'objectif de ce programme est d'établir des statistiques sur l'utilisation des différents modes de paiement effectués sur un compte bancaire. Nous supposons que les moyens techniques pour débiter un compte sont au nombre de trois : la Carte Bleue, le chéquier et le virement. Pour évaluer le taux d'utilisation de ces trois moyens de paiement, nous devons calculer le pourcentage d'utilisation de chaque technique par rapport aux autres. Par exemple, pour connaître le pourcentage d'utilisation de la Carte Bleue, nous utilisons le calcul suivant :

Nombre de paiements par Carte Bleue / Nombre total de paiements * 100

Liste des opérations

Partant du principe de décomposition d'un problème en sous-tâches plus simples à réaliser, distinguons, pour résoudre la question, les quatre actions suivantes :

1. Déterminer le nombre de débits par Carte Bleue, chèque et virement. Comme il s'agit du premier programme concernant ce thème, nous n'avons pas encore saisi de valeur, ni de ligne comptable. C'est pourquoi nous demandons à l'utilisateur de communiquer au programme ces trois informations, par l'intermédiaire du clavier.
2. Calculer le nombre total de paiements effectués.
3. Calculer le pourcentage d'utilisation de la Carte Bleue, du chéquier et du virement.
4. Afficher l'ensemble des résultats.

Copyright © 2016 Eyrolles.

Dans un premier temps, nous traiterons séparément chacun de ces points afin de les analyser entièrement. Pour finir, nous écrirons le programme dans son intégralité, en regroupant chacun des points étudiés.

1. Il s'agit d'écrire les instructions qui permettent à l'utilisateur de communiquer des informations à l'ordinateur à l'aide du clavier. Nous avons vu, au chapitre introductif, un exemple de saisie d'une valeur au clavier (voir section « Calcul de la circonférence d'un cercle »). Cette opération se réalise en deux temps : d'abord l'affichage à l'écran d'un message informant l'utilisateur d'une demande de saisie de valeur, puis la saisie effective de l'information. Pour notre problème, ces deux points se traduisent de la façon suivante :

```
Scanner lectureClavier = new Scanner(System.in);
System.out.print(" Nombre de paiements par Carte Bleue ") ;
nbCB = lectureClavier.nextInt();
System.out.print(" Nombre de cheques émis ") ;
nbCheque = lectureClavier.nextInt();
System.out.print(" Nombre de virements automatiques ") ;
nbVirement = lectureClavier.nextInt();
```

Chaque appel de la fonction `System.out.print()` affiche à l'écran le message placé entre `guillemets`. Trois messages sont affichés, chacun indiquant respectivement à quel mode de paiement est associée la valeur saisie par l'utilisateur.

Les valeurs à saisir correspondent aux nombres de débits dans chaque mode de paiement. Ces valeurs sont de type entier. La fonction `lectureClavier.nextInt()` donne l'ordre à l'ordinateur d'attendre la saisie d'une valeur entière. La saisie est effective lorsque l'utilisateur valide sa réponse en appuyant sur la touche `Entrée` du clavier. Trois valeurs sont à saisir, et il est nécessaire d'appeler trois fois la fonction `lecture Clavier.nextInt()`.

Pour en savoir plus Pour plus d'informations sur la fonction `lectureClavier.nextInt();`, voir le chapitre 2, « Communiquer une information ».

Une fois saisie, chaque valeur doit être stockée dans un emplacement mémoire distinct. Ces emplacements mémoire correspondent aux trois variables `nbCB`, `nbCheque` et `nbVirement` et sont déclarés en début de programme grâce à l'instruction :

```
int nbCB = 0, nbCheque = 0, nbVirement = 0 ;
```

2. Pour calculer le nombre total de paiements effectués, il suffit de faire la somme de toutes les opérations de débit pour tous les types de paiement, soit l'instruction :

```
nbDebit = nbCB + nbCheque + nbVirement ;
```

La variable `nbDebit` permet la mémorisation du nombre total d'opérations effectuées, quel que soit le mode de paiement. Elle doit être déclarée en même temps que les autres variables du même type :

```
int nbCB = 0, nbCheque = 0, nbVirement = 0, nbDebit = 0 ;
```

Copyright © 2016 Eyrolles.

3. Pour calculer le pourcentage d'utilisation de la Carte Bleue, du chéquier et du virement, nous allons d'abord étudier le mode Carte Bleue puis appliquer cette analyse aux autres modes de paiement. Rappelons que la formule du calcul de pourcentage pour la Carte Bleue est :

   Nombre de paiements par Carte Bleue / Nombre total de paiements * 100

   soit, en utilisant les variables déclarées au point 1 : `nbCB / nbDebit * 100`.

   Examinons sur un exemple numérique le résultat d'un tel calcul. Supposons pour cela que nous ayons effectué 10 retraits Carte Bleue sur un total de 40 retraits. Nous obtenons le calcul suivant : `10 / 40 * 100`. Soit `0 * 100`, c'est-à-dire `0`. La division est la première opération exécutée parce qu'elle est du même groupe que la multiplication et qu'elle apparaît en premier dans l'opération. De surcroît, les valeurs étant de type entier, la division a pour résultat un nombre entier. Ici `10/40` a pour résultat `0`.

   Pour corriger cette erreur de calcul, l'idée est de réaliser une division sur des valeurs réelles et non sur des entiers. Pour cela, nous utilisons le mécanisme du cast, qui, placé devant la variable `nbCB`, transforme cette dernière en variable de type réel et permet la division en réel. Pour stocker le résultat de cette opération, nous déclarons une variable de type `float`, nommée `prctCB`.

   L'instruction :

   ```
   prctCB = (float) nbCB / nbDebit * 100 ;
   ```

   permet de trouver un résultat cohérent. Vérifions cela sur un exemple numérique. Supposons que nous ayons effectué 10 débits par Carte Bleue sur un total de 20 retraits. Grâce au cast, la valeur `10` correspondant à `nbCB` est transformée en `10.0`. La division par 20 a donc un résultat réel égal à 0.5. Le taux d'utilisation de la Carte Bleue est donc de `0.5 * 100`, soit 50 %.

   Pour établir le pourcentage relatif aux modes chéquier et virement, il suffit d'appliquer le même calcul, en utilisant des variables appropriées aux deux autres moyens de paiement. En nommant `prctCh` et `prctVi` les variables associées aux modes de paiement par chèque et par virement automatique, le taux d'utilisation pour chacun de ces modes s'écrit :

   ```
   prctCh = (float) nbCheque / nbDebit * 100 ;
   prctVi = (float) nbVirement / nbDebit * 100 ;
   ```

4. L'affichage des résultats s'effectue par l'intermédiaire de la fonction `System.out.print()`. Les valeurs calculées sont commentées de la façon suivante :

   ```
   System.out.println(" Vous avez emis " + nbDebit + " ordres de
   ➥debit ") ;
   System.out.println(" dont " + prctCB + " % par Carte Bleue ") ;
   System.out.println("       " + prctCh + " % par cheque ") ;
   System.out.println("       " + prctVi + " % par virement ") ;
   ```

Le programme final s'écrit en regroupant l'ensemble des instructions définies précédemment et en les insérant dans une classe à l'intérieur de la fonction `main()`.

Copyright © 2016 Eyrolles.

Le code source complet

```
import java.util.*;
public class Statistique
{
  public static void main (String [] arg)
  {
   int nbCB = 0, nbCheque = 0, nbVirement = 0, nbDebit = 0 ;
   float prctCB, prctCh, prctVi ;
   Scanner lectureClavier = new Scanner(System.in);
   System.out.print(" Nombre de paiements par Carte Bleue : ") ;
   nbCB = lectureClavier.nextInt();
   System.out.print(" Nombre de cheques emis : ") ;
   nbCheque = lectureClavier.nextInt();
   System.out.print(" Nombre de virements automatiques : ") ;
   nbVirement = lectureClavier.nextInt();

   nbDebit = nbCB + nbCheque + nbVirement;

   prctCB = (float) nbCB / nbDebit * 100 ;
   prctCh = (float) nbCheque / nbDebit * 100 ;
   prctVi = (float) nbVirement / nbDebit * 100 ;

  System.out.println("Vous avez emis " + nbDebit + " ordres de
                      debit") ;
  System.out.println("dont " + prctCB + " % par Carte Bleue") ;
  System.out.println("     " + prctCh + " % par cheque") ;
  System.out.println("     " + prctVi + " % par virement") ;
  }
}
```

Résultat de l'exécution

À l'exécution de ce programme, nous avons à l'écran l'affichage suivant (les caractères grisés sont des valeurs choisies par l'utilisateur) :

```
Nombre de paiements par Carte Bleue : 5
Nombre de cheques emis : 10
mbre de virements automatiques : 5
Vous avez emis 20 ordres de debit
dont 25.0 % par Carte Bleue
     50.0 % par cheque
     25.0 % par virement
```

Copyright © 2016 Eyrolles.

Résumé

Une **variable** est caractérisée par un **nom** et un **type**. Le nom sert à repérer un emplacement mémoire. Le type détermine la taille de cet emplacement, ainsi que la manière dont l'information est codée, les opérations autorisées et l'intervalle des valeurs représentables.

Il existe plusieurs types simples, dont les plus utilisés sont les suivants :

- `int`. Représente les entiers variant, pour le langage Java, entre – 2 147 483 648 et 2 147 483 647.
- `double`. Décrit de manière approchée les nombres réels dont la valeur absolue est grande. Les variables de type double se notent soit sous forme décimale (67.7, –9.2, 0.48 ou .22), soit sous forme exponentielle 3.14E4, .325707e2, –45.567E-5.
- `char`. Désigne les caractères. Les valeurs de type caractère se notent en plaçant entre apostrophes le caractère lui-même.

L'instruction d'**affectation** permet de placer une valeur dans une variable. Elle est de la forme : `variable = expression;` .

Elle calcule d'abord la valeur de l'expression mentionnée à droite du signe = , puis elle l'affecte à la variable placée à gauche du signe.

Il est conseillé d'attribuer une valeur initiale à une variable au moment de sa déclaration. Par exemple `int i = 6;` ou `char c = 'n';`.

Pour calculer des expressions mathématiques, il existe cinq **opérateurs** arithmétiques : `+ - * / %`.

Ces opérateurs sont utilisés respectivement pour l'addition, la soustraction, la multiplication, la division et le modulo (reste de la division entière). Les expressions arithmétiques sont calculées à partir des règles suivantes :

- Entier `+ - * / %` entier donne un entier.
- Réel `+ - * /` réel donne un réel.
- Les opérations mixtes du type :

entier `+ - * /` réel ou réel `+ - * /` entier

donnent un résultat dans la mesure où la valeur résultante n'est pas dénaturée par la conversion des types. Les conversions sont effectuées automatiquement dans le sens suivant :

```
byte -> short -> int -> long -> float -> double
```

Un `int` peut donc être transformé en un `double`. L'inverse n'est possible que lorsque le mode de conversion est explicitement décrit dans l'expression, comme dans `n = (int) x`, où `n` est de type `int` et `x` de type `double`.

L'information ainsi transformée est tronquée pour être codée sur moins d'octets.

- Il existe des règles de **priorité** entre les opérateurs. Pour cela, deux groupes d'opérateurs sont définis.

Groupe 1	Groupe 2
`+ -`	`* / %`

Dans un même groupe, l'opération se fait dans l'ordre d'apparition des opérateurs.

Le second groupe a priorité sur le premier.

Les parenthèses permettent la modification des priorités.

Copyright © 2016 Eyrolles.

Exercices

Repérer les instructions de déclaration, observer la syntaxe d'une instruction

Exercice 1.1 Observez ce qui suit, et indiquez ce qui est ou n'est pas une déclaration et ce qui est ou n'est pas valide :

```
a.  int i, j, valeur ;
b.  limite - j = 1024 ;
c.  val = valeur / 16 ;
d.  char char ;
j + 1 ;
f.  int X ;
g.  float A ;
h.  A = X / 2 ;
i.  X = A / 2 ;
j.  X = X / 2 ;
```

Comprendre le mécanisme de l'affectation

Exercice 1.2 Quelles sont les valeurs des variables A, B, C après l'exécution de chacun des extraits de programme suivants :

a.	b.
`float A = 3.5f ;`	`double A = 0.1 ;`
`float B = 1.5f ;`	`double B = 1.1 ;`
`float C ;`	`double C, D ;`
`C = A + B ;`	`B = A ;`
`B = A + C ;`	`C = B ;`
`A = B ;`	`D = C ;`
	`A = D ;`

Copyright © 2016 Eyrolles.

Exercice 1.3 Quelles sont les valeurs des variables a, b, valeur, x, y et z, après l'exécution de chacune des instructions suivantes :

a.	b.	c.

`int a = 5, b ;`
`b = a + 4 ;`
`a = a + 1 ;`
`b = a - 4 ;`
	`int valeur = 2 ;`
	`valeur = valeur + 1 ;`
	`valeur = valeur * 2 ;`
	`valeur = valeur % 5 ;`
		`int x = 2, y = 10, z ;`
		`z = x + y ;`
		`x = 5 ;`
		`z = z - x ;`

Comprendre le mécanisme d'échange de valeurs

Exercice 1.4 Dans chacun des cas, quelles sont les valeurs des variables a et b après l'exécution de chacune des instructions suivantes :

1.	2.

`int a = 5 ;`
`int b = 7 ;`
`a = b ;`
`b = a ;`
	`int a = 5 ;`
	`int b = 7 ;`
	`b = a ;`
	`a = b ;`

Exercice 1.5 Laquelle des options suivantes permet d'échanger les valeurs des deux variables a et b ?

```
a = b ; b = a ;
t = a ; a = b ; b = t ;
t = a ; b = a ; t = b ;
```

Exercice 1.6 Soit trois variables a, b et c (entières). Écrivez les instructions permutant les valeurs, de sorte que la valeur de a passe dans b, celle de b dans c et celle de c dans a. N'utilisez qu'une (et une seule) variable entière supplémentaire, nommée tmp.

Exercice 1.7 Quel est l'effet des instructions suivantes sur les variables a et b (pour vous aider, initialisez a à 2 et b à 5) :

```
a = a + b ;
b = a - b ;
a = a - b ;
```

Copyright © 2016 Eyrolles.

Calculer des expressions mixtes

Exercice 1.8 Donnez les valeurs des expressions suivantes, sachant que `i` et `j` sont de type `int` et `x` et `y` de type `double` (`x` = 2.0, `y` = 3.0) :

```
a.  i = 100 / 6 ;
b.  j = 100 % 6 ;
c.  i = 5 % 8  ;
d.  (3 * i - 2 * j) / (2 * x - y) ;
e.  2 * ((i / 5) + (4 * (j - 3)) % (i + j - 2)) ;
f.  (i - 3 * j) / (x + 2 * y) / (i - j) ;
```

Exercice 1.9 Donnez le type et la valeur des expressions suivantes, sachant que `n`, `p`, `r`, `s` et `t` sont de type `int` (`n` = 10, `p` = 7, `r` = 8, `s` = 7, `t` = 21) et que `x` est de type `float` (`x = 2.0f`) :

a.	b.
`x + n % p`	`r + t / s`
`x + n / p`	`( r + t ) / s`
`(x + n) / p`	`r + t % s`
`5. * n`	`(r + t) % s`
`(n + 1) / n`	`r + s / r + s`
`(n + 1.0) / n`	`(r + s) / ( r + s)`
`r + s / t`	`r + s % t`

Comprendre le mécanisme du cast

Exercice 1.10 Soit les déclarations suivantes :

```
int valeur = 7, chiffre = 2, i1, i2 ;
float f1, f2 ;
```

Quelles sont les valeurs attribuées à `i1`, `i2`, `f1` et `f2` après le calcul de :

```
i1 = valeur / chiffre ;
i2 = chiffre / valeur ;
f1 = (float) (valeur / chiffre) ;
f2 = (float) (valeur / chiffre) + 0.5f ;
i1 = (int) f1 ;
i2 = (int) f2 ;
f1 = (float) valeur / (float) chiffre ;
f2 = (float) valeur / (float) chiffre + 0.5f ;
i1 = (int) f1 ;
i2 = (int) f2 ;
```

Copyright © 2016 Eyrolles.

Le projet : Gestion d'un compte bancaire

Déterminer les variables nécessaires au programme

Le programme de gestion d'un compte bancaire ne peut s'écrire et s'exécuter sans aucune variable. Pour pouvoir définir toutes les variables nécessaires à la bonne marche du programme, nous devons examiner attentivement le cahier des charges décrit au chapitre introductif, « Naissance d'un programme ».

La section « Les objets manipulés » nous donne une première idée des variables à déclarer. Toutes les données relatives au compte bancaire y sont décrites.

Un compte bancaire est défini par un ensemble de données :

- un numéro de compte ;
- un type de compte (courant, épargne, joint, etc.) ;
- des lignes comptables possédant chacune une valeur, une date, un thème et un moyen de paiement.

Ces données peuvent être représentées de la façon suivante :

Données	Exemple	Type de l'objet
Numéro du compte	4010.205.530	Suite de caractères
Type du compte	Courant	Suite de caractères
Valeur	–1520.30	Numérique
Date	04 03 1978	Date
Thème	Loyer	Suite de caractères
Moyen de paiement	CB	Suite de caractères

Compte tenu de ces informations, donnez un nom et un type Java pour chaque donnée définie ci-dessus.

Remarquons que le type qui représente les suites de caractères (`String`) n'a pas encore été étudié, ni toutes ses fonctionnalités. Il est possible de transformer pour l'instant les données `Type du compte`, `Thème` et `Moyen de paiement` en caractères simples. Par exemple, le caractère `C` caractérise le type du compte `Courant`, le caractère `J` le compte `Joint` et le caractère `E` le compte `Epargne`.

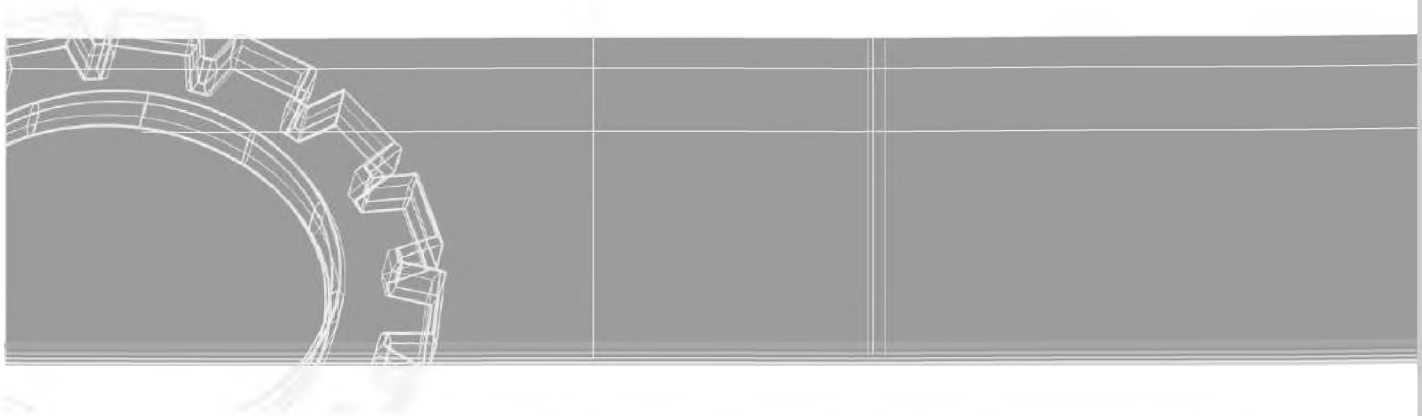
De la même façon, la donnée `Numéro du compte` peut être transformée dans un premier temps en type `long`.

Copyright © 2016 Eyrolles.

Copyright © 2016 Eyrolles.

Chapitre 2

Communiquer une information



Un programme n'a d'intérêt que s'il produit un résultat. Pour communiquer ce résultat, l'ordinateur utilise l'écran. Cette action, qui consiste à afficher un message, est appelée opération de **sortie**, ou d'écriture, de données.

Parallèlement, un programme ne produit de résultats que si l'utilisateur lui fournit au préalable des informations. Ces informations, ou données, sont transmises au programme le plus souvent par l'intermédiaire d'un clavier. Dans le jargon informatique, cette opération est appelée opération de saisie, d'**entrée** ou encore de lecture de données.

Dans ce chapitre, nous commençons par étudier les fonctionnalités proposées par le langage Java pour gérer les opérations d'entrée-sortie (voir section « La bibliothèque System »).

À la section « L'affichage de données », nous examinons ensuite comment afficher à l'écran des messages et des données. Enfin, à la section « La saisie de données », nous proposons une technique de saisie de valeurs au clavier.

La bibliothèque System

Nous l'avons vu dans les exemples des chapitres précédents, l'affichage de valeurs ou de texte est réalisé par l'utilisation d'une fonction prédéfinie du langage Java. Cette fonction a pour nom d'appel `System.out.print()`.

Copyright © 2016 Eyrolles.

Pourquoi un nom si complexe, pour réaliser une action aussi « simple » que l'affichage de données ?

Le langage Java est accompagné d'un ensemble de bibliothèques de programmes préécrits, qui épargnent au programmeur d'avoir à réécrire ce qui a déjà été fait depuis les débuts de l'ère informatique. Ces bibliothèques portent chacune un nom qui renseigne sur leur fonctionnalité. Ainsi, la bibliothèque où se trouve l'ensemble des fonctions de calcul mathématique s'appelle `Math`, et celle relative à la gestion des éléments de bas niveau (écran, clavier, etc.) impliquant le système de l'ordinateur s'appelle `System`.

La gestion de l'affichage d'un message à l'écran ou la saisie de valeurs au clavier fait partie des fonctions impliquant le système de l'ordinateur. C'est pourquoi le nom d'appel de telles fonctions a pour premier terme `System`.

Les opérations d'entrée ou de sortie de données impliquent le système de l'ordinateur mais sont en rapport inverse l'une de l'autre. Pour dissocier ces opérations, la bibliothèque `System` est composée de deux sous-ensembles, `in` et `out`. L'affichage est une opération de sortie et fait donc partie des éléments `out` de la classe `System`. Le point (`.`) qui relie le mot `System` à `out` permet d'expliquer à l'ordinateur que l'on souhaite accéder au sous-ensemble `out` de la bibliothèque `System` plutôt qu'au sous-ensemble `in`. Pour finir, nous faisons appel, dans le sous-ensemble `out`, à la fonction `print()`, qui affiche un message à l'écran. Le nom de la fonction `print()` signifie imprimer, car, au tout début de l'informatique, les ordinateurs n'avaient pas d'écran, et les résultats d'un calcul étaient imprimés sur papier ou sur carte informatique.

La notation point (`.`) est une écriture courante en programmation objet. Comme nous le verrons au chapitre 7, « Les classes et les objets », elle offre le moyen d'accéder à des programmes ou à des données spécifiques.

Notons que, dans la classe `System`, se trouve aussi le sous-ensemble `err`, qui permet d'afficher les erreurs éventuelles d'un programme sur la sortie standard des erreurs. Ce type de sortie n'est défini que dans le monde Unix, et la sortie `err` est identique à la sortie `out` dans le monde DOS.

L'affichage de données

Le principe général, pour l'affichage d'un message, est de placer ce dernier en paramètre de la fonction `System.out.print()`, c'est-à-dire à l'intérieur des parenthèses qui suivent le terme `System.out.print`. Plusieurs possibilités existent quant à la forme et à la syntaxe de ce message, et nous les présentons ci-après.

Copyright © 2016 Eyrolles.

Affichage de la valeur d'une variable

Soit la variable entière `valeur`. L'affichage de son contenu à l'écran est réalisé par :

```
int valeur = 22 ;
System.out.print(valeur) ;
```

À l'écran, le résultat s'affiche ainsi :

```
22
```

Affichage d'un commentaire

Le fait d'écrire une valeur numérique, sans autre commentaire, n'a que peu d'intérêt. Pour expliquer un résultat, il est possible d'ajouter du texte avant ou après la variable, comme dans l'exemple :

```
System.out.print(" Le montant s'eleve a : " + valeur) ;
```

ou

```
System.out.print(valeur + " correspond au montant total ") ;
```

Pour ajouter un commentaire avant ou après une variable, il suffit de le placer entre guillemets (`" "`) et de l'accrocher à la variable à l'aide du signe +. De cette façon, le compilateur est capable de distinguer le texte à afficher du nom de la variable. Tout caractère placé entre guillemets est un message, alors qu'un mot non entouré de guillemets correspond au nom d'une variable.

Question En reprenant la même variable `valeur` qu'à l'exemple précédent, quel est le résultat affiché par les instructions précédentes ?

Réponse La première instruction affiche à l'écran :

```
Le montant s'eleve a : 22
```

La seconde :

```
22 correspond au montant total
```

Affichage de plusieurs variables

On peut afficher le contenu de plusieurs variables en utilisant la même technique. Les commentaires sont placés entre guillemets, et les variables sont précédées, entourées ou suivies du caractère +. Le signe + réunit chaque terme de l'affichage au suivant ou au précédent. Pour afficher le contenu de deux variables :

```
int v = 5, s = 33 ;
```

Copyright © 2016 Eyrolles.

nous écrivons

```
System.out.print(v + " elements valent au total " + s + " euros ") ;
```

Question Quel est le résultat de l'instruction précédente ?

Réponse L'exécution de cette instruction a pour résultat :

```
5 elements valent au total 33 euros
```

Affichage de la valeur d'une expression arithmétique

Dans une instruction d'affichage, il est possible d'afficher directement le résultat d'une expression mathématique, sans qu'elle ait été calculée auparavant. Par exemple, nous pouvons écrire :

```
int a = 10, b = 5 ;
System.out.print(a+" fois " + b + " est egal a " + a * b) ;
```

À l'écran, le résultat s'affiche ainsi :

```
10 fois 5 est egal a 50
```

Mais attention ! Cette expression est calculée au cours de l'exécution de l'instruction, elle n'est pas mémorisée dans un emplacement mémoire. Le résultat ne peut donc pas être réutilisé dans un autre calcul.

L'écriture d'une expression mathématique à l'intérieur de la fonction d'affichage peut être source de confusion pour le compilateur, surtout si l'expression mathématique comporte un ou plusieurs signes +. En remplaçant, dans l'exemple précédent, le signe * par +, nous obtenons :

```
int a = 10, b = 5 ;
System.out.print(a + " plus " + b + " est egal a " + a + b) ;
```

À l'écran, le résultat s'affiche de la façon suivante :

```
10 plus 5 est egal a 105
```

Remarque L'ordinateur ne peut pas afficher la somme de a et de b parce que, lorsque le signe + est placé dans la fonction d'affichage, il a pour rôle de réunir des valeurs et du texte sur une même ligne d'affichage, et non d'additionner deux valeurs. 105 n'est que la réunion de 10 et de 5. On dit qu'il s'agit d'une opération de **concaténation**.

Pour afficher le résultat d'une addition, il est nécessaire de placer entre parenthèses le calcul à afficher. Par exemple :

```
int a = 10, b = 5 ;
System.out.print(a + " plus " + b + " est egal a " + (a+b)) ;
```

Copyright © 2016 Eyrolles.

Le résultat à l'écran est :

```
10 plus 5 est egal a 15
```

Affichage d'un texte

Nous pouvons aussi afficher un simple texte sans utiliser de variable :

```
System.out.print("Qui seme le vent recolte la tempete ! ") ;
```

À l'écran, le résultat s'affiche ainsi :

```
Qui seme le vent recolte la tempete !
```

Pour changer de ligne

Signalons que l'instruction `System.out.print` affiche les informations à la suite de celles qui ont été affichées par un précédent `System.out.print`. Il n'y a pas de passage à la ligne entre deux instructions d'affichage. Ainsi, les instructions :

```
System.out.print("Qui seme le vent ") ;
System.out.print("recolte la tempete ! ") ;
```

ont le même résultat à l'écran que celle de l'exemple précédent :

```
Qui seme le vent recolte la tempete !
```

Pour obtenir un passage à la ligne, il est nécessaire d'utiliser la fonction

```
System.out.println()
```

Ainsi, les instructions :

```
System.out.println("Qui seme le vent ") ;
System.out.print("recolte la tempete ! ") ;
```

ont pour résultat :

```
Qui seme le vent
recolte la tempete !
```

Les caractères spéciaux

La table Unicode définit tous les caractères textuels (alphanumériques) et semi-graphiques (idéogrammes, etc.).

Les caractères spéciaux sont définis à partir du code-point `0080` de la table Unicode. Ils correspondent à des caractères n'existant pas sur le clavier mais qui sont néanmoins utiles. Les caractères accentués font aussi partie des caractères spéciaux, les claviers Qwerty américains ne possédant pas ce type de caractères.

Copyright © 2016 Eyrolles.

Pour afficher un message comportant des caractères n'existant pas sur le clavier, ou comprenant des caractères accentués, vous devez insérer à l'intérieur du message le code Unicode des caractères souhaités.

Extension Web La table Unicode est décrite au chapitre 1, « Stocker une information », à la section « Les types de base en Java – Catégorie caractère ». Vous pouvez également consulter la table Unicode des caractères de l'alphabet latin, en ouvrant les fichiers `Unicode0000a007F.pdf` et `Unicode0080a00FF.pdf` placés sur l'extension Web de l'ouvrage.

Ainsi le proverbe `"Qui sème le vent récolte la tempête"` s'écrit en Java `"Qui s\u00E8me le vent, r\u00E9colte la temp\u00EAte"`.

Il est à noter que même si le langage Java utilise le jeu de caractères Unicode, le traitement des chaînes de caractères par une application Java dépend du jeu de caractères par défaut du système d'exploitation ou de l'environnement de développement de l'application, ce qui pose parfois problème pour l'affichage des caractères accentués.

Par exemple, dans le jeu de caractères ANSI utilisé par Windows, le caractère ù se trouve à la même position que le caractère tréma (¨) dans le jeu de caractères DOS. Ainsi, le mot `"où"` s'affiche `"où"` avec l'outil Bloc-notes de Windows et `"ö"` avec l'éditeur edit de DOS.

Pour éviter d'obtenir des caractères plus ou moins étranges à l'affichage d'une chaîne de caractères, la solution consiste à encoder la chaîne de caractères dans le système d'encodage par défaut de l'environnement du système utilisé. Cette technique est réalisée par le programme suivant :

```
public class EncodageParDefaut {
    public static void main(String[] args)
            throws java.io.IOException {
  String encodage = System.getProperty("file.encoding"); // ❶
  System.out.println("Encodage par défaut : " + encodage);
  String proverbe = "Qui s\u00E8me le vent, r\u00E9colte la
                     temp\u00EAte " ;                        // ❷
  String proverbeEncode = new String (proverbe.getBytes(),
  ➥encodage );                                              // ❸
   System.out.println(" proverbe : " + proverbeEncode );
 }
}
```

❶ La méthode `System.getProperty()` récupère le système d'encodage par défaut de votre environnement de travail lorsque le terme `"file.encoding"` est passé en paramètre de la méthode.

❷ La chaîne de caractères `"Qui sème le vent, récolte la tempête"` est mémorisée dans la variable `proverbe`, en utilisant les codes Unicode des caractères è, é et ê, respectivement.

Copyright © 2016 Eyrolles.

❸ L'instruction `new String(...)` est un peu plus complexe à déchiffrer. Elle s'exécute en trois temps :

- Le terme `proverbe.getBytes()` transforme la chaîne de caractères enregistrée dans `proverbe`, en une suite d'octets.
- L'expression `new String(..., encodage)` crée une nouvelle chaîne de caractères à partir de la suite d'octets passée en premier paramètre, selon l'encodage fourni en second paramètre. Ici, l'encodage est celui de l'environnement dans lequel vous travaillez.
- La nouvelle chaîne ainsi créée est enregistrée dans la variable `proverbeEncode` grâce à l'opérateur =.

Remarque

Notez l'expression `throws IOException` placée juste après l'en-tête de la fonction `main()`. La présence de cette expression indique au compilateur que la méthode `main()` est susceptible de traiter ou de propager une éventuelle erreur du type `IOException`, qui pourrait apparaître en cours d'exécution. Ainsi, une erreur de type `UnsupportedEncodingException` est propagée lorsque l'interpréteur Java ne connaît pas l'encodage par défaut de l'environnement dans lequel vous travaillez.

Pour en savoir plus

Pour plus de précision sur la notion d'exception, voir la section « Gérer les exceptions », à la fin du chapitre 10, « Collectionner un nombre indéterminé d'objets » et l'exercice 10.8 de ce même chapitre. La classe `String` et l'opérateur new sont étudiés plus précisément au chapitre 7, « Les classes et les objets ».

Exécution sous DOS

L'exécution du programme `EncodageParDefaut` a pour résultat d'afficher, dans la fenêtre de commandes DOS :

```
Encodage par défaut : Cp1252
Proverbe : Qui sème le vent, récolte la tempête
```

Il se peut cependant que l'affichage ne soit pas encore tout à fait correct. Pour cela, vous devez vérifier que l'encodage par défaut de la fenêtre `cmd.exe` soit bien `Cp1252`.

Remarque

Le code page `cp1252` est le jeu de caractères par défaut de Windows. Il correspond à la norme ISO-8859-1. La fenêtre `cmd.exe` utilise en général le code page `IBM850` qui n'est pas supporté par Java.

Pour modifier le jeu de caractères par défaut de la fenêtre `cmd.exe`, vous devez taper la commande `chcp` dans la fenêtre de commandes afin de vérifier quel jeu de caractères a été chargé.

Si la réponse est autre chose que :

```
Page de codes active : 1252
```

Copyright © 2016 Eyrolles.

il convient de modifier les pages de code par défaut, en tapant la commande :

```
chcp 1252
```

La police de caractères utilisée par la fenêtre de commandes a également une incidence sur l'affichage des caractères accentués. Pour afficher correctement les accents, vous devez modifier la police de caractères par défaut de la fenêtre de commandes. Pour cela :

- Cliquer droit sur la barre d'en-tête de la fenêtre de commandes.
- Sélectionner l'item `Propriétés`, dans le menu contextuel qui apparaît en effectuant un clic droit.
- Cliquer sur l'onglet `Police` et choisir la police de caractères `Lucida Console`.
- Valider le tout en cliquant sur le bouton `OK`.

Exécution sous Linux

Sous Linux (distribution Debian), le programme `EncodageParDefaut` a pour résultat :

```
Encodage par défaut : ISO-8859-15
Proverbe : Qui sème le vent, récolte la tempête
```

La saisie de données

Java est un langage conçu avant tout pour être exécuté dans un environnement Internet et utilisant des programmes essentiellement axés sur le concept d'interface graphique (gestion des boutons, menus, fenêtres, etc.). Dans ce type d'environnement, la saisie de données est gérée par des fenêtres spécialisées, appelées fenêtres de dialogue.

L'objectif de cet ouvrage est d'initier le lecteur au langage Java et, surtout, de lui faire comprendre comment construire et élaborer un programme. Pour cet apprentissage (algorithme et langage), il n'est pas recommandé de se lancer dans l'écriture de programmes utilisant des boutons, des menus et autres fenêtres sans avoir étudié au préalable toute la bibliothèque AWT (*Abstract Windowing Toolkit*) de Java. Cette bibliothèque facilite, il est vrai, la construction d'applications graphiques, mais elle complique et alourdit l'écriture des programmes.

Pour en savoir plus Pour plus de détails sur la bibliothèque graphique AWT, reportez-vous au chapitre 11, « Dessiner des objets ».

C'est pourquoi nous avons délibérément choisi de travailler dans un environnement non graphique, plus simple à programmer.

Dans cet environnement, le langage Java propose la fonction `System.in.read()`, qui permet la saisie de données au clavier, sans l'intermédiaire de fenêtres graphiques. Cette fonction est définie dans la bibliothèque `System`, à l'intérieur du sous-ensemble `in`. Elle utilise le programme de lecture au clavier `read()`.

Copyright © 2016 Eyrolles.

La fonction `System.in.read()` permet de récupérer un et un seul caractère saisi au clavier. Si l'utilisateur souhaite saisir des valeurs ou des noms composés de plusieurs caractères, le programme doit contenir autant d'instructions `System.in.read()` que de caractères à saisir. Le nombre de caractères à saisir variant suivant l'utilisation de l'application, cette fonction n'est pas directement utilisable de cette façon.

La classe Scanner

Avant la version Java 1.5, il était difficile de saisir une valeur au clavier (voir section « Saisir un nombre entier au clavier », chapitre 4, « Faire des répétitions »). Aujourd'hui la situation s'est améliorée grâce à la classe `Scanner`, du package `java.util`.

La classe `Scanner` propose au lecteur un ensemble de fonctions de lecture qui permet de saisir autant de caractères que souhaité. Pour terminer la saisie, il suffit de la valider en appuyant sur la touche `Entrée` du clavier. De plus, il existe autant de fonctions de lecture que de types de variables. Il est très facile de saisir des valeurs numériques de type entier (`byte`, `short`, `int` et `long`) ou réel (`float` et `double`) et des caractères de type `char` ou `String`.

Pour ce faire, la technique consiste à :

1. importer la classe `Scanner` grâce à l'instruction :

```
import java.util.*;
```

Cette instruction doit être placée en tout début de programme, avant la définition de la classe. Elle est utilisée pour préciser au compilateur qu'il doit charger les classes enregistrées dans le paquetage `java.util`.

Remarque Comme son nom l'indique, le package `java.util` rassemble différents outils de programmation très utiles. On y trouve par exemple la classe `Date` qui permet de connaître l'heure à la seconde prêt, ou encore les classes `Vector` ou `HashTable` qui offre des outils très performants pour traiter des listes de données (voir chapitre 10, « Collectionner un nombre indéterminé d'objets »).

2. créer un objet de type `Scanner` à l'aide de l'instruction :

```
Scanner lectureClavier = new Scanner(System.in);
```

L'objet se nomme `lectureClavier`. Il est créé grâce à l'opérateur `new`. Le fait de placer en paramètre le terme `System.in` indique au compilateur, que l'objet `lectureClavier` doit scanner (en français, parcourir ou encore balayer) le système d'entrée des valeurs, c'est-à-dire le clavier.

Pour en savoir plus La notion d'objet ainsi que l'opérateur `new` sont étudiés au chapitre 7, « Les classes et les objets ».

Copyright © 2016 Eyrolles.

3. utiliser une méthode de la classe `Scanner` pour lire un entier, un réel ou encore un caractère. Ainsi par exemple la méthode `nextInt()` appliquée à l'objet `lectureClavier` permet la saisie d'une valeur de type `int`.

L'exemple ci-après regroupe l'ensemble des méthodes permettant la saisie de valeurs de type `short`, `byte`, `int`, `long`, `float`, `double`, `char` et `String`.

Exemple : code source complet

```
import java.util.*;
public class TestLectureClavier {
 public static void main (String [] Arg) {
   int intLu;
   float floatLu;
   double doubleLu;
   char charLu;
   byte byteLu;
   long longLu;
   short shortLu;
   String stringLu;
   Scanner lectureClavier = new Scanner(System.in);

   System.out.println("Entrez un short : ");
   shortLu = lectureClavier.nextShort();
   System.out.println("Entrez un byte : ");
   byteLu = lectureClavier.nextByte();
   System.out.println("Entrez un int : ");
   intLu = lectureClavier.nextInt();
   System.out.println("Entrez un long : ");
   longLu = lectureClavier.nextLong();
   System.out.println("Entrez un float : ");
   floatLu = lectureClavier.nextFloat();
   System.out.println("Entrez un double : ");
   doubleLu = lectureClavier.nextDouble();
   System.out.println("Entrez un String: ");
   stringLu = lectureClavier.next();
   System.out.println("Entrez un char : ");
   charLu = lectureClavier.next().charAt(0);
   System.out.println("entier : " + intLu);
   System.out.println("float : " + floatLu);
   System.out.println("double : " + doubleLu);
   System.out.println("char : " + charLu);
```

Copyright © 2016 Eyrolles.

```
        System.out.println("byte : " + byteLu);
        System.out.println("short : " + shortLu);
        System.out.println("String : " + stringLu);
        System.out.println("long : " + longLu);
    }
}
```

Après la déclaration des variables, le programme demande la saisie de valeurs d'un certain type. L'utilisateur fournit la valeur correspondant au type demandé et valide la saisie en appuyant sur la touche `Entrée` du clavier. Une fois saisies, les valeurs sont affichées à l'écran.

Observez la particularité de l'instruction de saisie d'un caractère : `next().charAt(0)`. Cette instruction demande de bien connaître la classe `String` (voir chapitre 7, « Les classes et les objets », section « La classe String, une approche de la notion d'objet »). Succinctement, l'instruction :

```
charLu = lectureClavier.next().charAt(0);
```

a pour rôle de saisir une suite de caractères grâce à la méthode `next()`, puis de ne retenir que le premier caractère de cette suite à l'aide de la méthode `charAt(0)`. De cette façon, même si l'utilisateur saisi plusieurs caractères, seul le premier saisi (numéroté `0`) est enregistré dans la variable `charLu`.

Remarque

La méthode `LectureClavier.next()` ne permet pas de saisir des phrases mais juste un mot. En effet, le caractère « Espace » constitue pour cette méthode, un marqueur de fin de saisie au clavier.

Pour saisir une phrase (c'est-à-dire une suite de mots séparés par des espaces), vous devez utiliser la méthode `LectureClavier.nextLine()`.

Question

Que se passe-t-il si l'utilisateur saisit la valeur `1.5` lorsque l'application `testLectureClavier` demande d'entrer un `float` ou un `double`.

Réponse

L'application cesse son exécution en précisant l'erreur `java.util.InputMismatchException`.

En effet, en France, un nombre réel s'écrit à l'aide d'une virgule alors qu'aux États-Unis, on utilise le point. En utilisant la classe `Scanner` sur un système d'exploitation réglé en zone française, nous devons saisir les valeurs réelles avec une virgule. Pour utiliser la notation américaine, nous devons modifier la localité grâce à l'instruction :

```
lectureClavier.useLocale(Locale.US);
```

L'instruction :

```
lectureClavier.useLocale(Locale.FRENCH);
```

permet de revenir à la saisie des valeurs réelles avec une virgule.

Copyright © 2016 Eyrolles.

Résultat de l'exécution

Les caractères grisés sont des valeurs choisies par l'utilisateur.

```
Entrez un byte : 100
Entrez un short :-30560
Entrez un int : 125698
Entrez un long : 98768765
Entrez un float : 3.14159
Entrez un double :123.876453097432
Entrez un String: Exemple
Entrez un char : A
vous avez entre le byte : 100
vous avez entre le short : -30560
vous avez entre l'entier : 125698
vous avez entre le long : 98768765
vous avez entre le float : 3,14159
vous avez entre le double : 123,876453097432
vous avez entre le caractere : A
vous avez entre le String : Exemple
```

Question Que réalisent les instructions suivantes ?

```
String pdt;
float prix;
int quantité;
Scanner lectureClavier = new Scanner(System.in);
System.out.print("Entrez le nom du produit : ");
pdt = lectureClavier.next();
System.out.print("Entrez le prix du produit : ");
prix = lectureClavier.nextDouble();
System.out.print("Entrez la quantite achetee: ");
quantité = lectureClavier.nextInt();
System.out.print("Vous avez achete : " + quantité + " " + pdt);
System.out.println(" au prix unitaire de "+ prix + " euros");
System.out.print("Montant total : "+ quantité * prix + " euros");
```

Réponse Les instructions précédentes réalisent l'affichage et la saisie de valeurs de la façon suivante :

```
Entrez le nom du produit : DVD
Entrez le prix du produit: 33.5
Entrez la quantite achetee: 2
Vous avez achete 2 DVD au prix unitaire de 33.5 euros
Montant total : 67.0 euros
```

Copyright © 2016 Eyrolles.

Résumé

Pour communiquer une information, l'ordinateur affiche un message à l'écran. On dit qu'il réalise une opération de **sortie** (out) ou d'écriture de données. À l'inverse, lorsque l'utilisateur communique des données au programme par l'intermédiaire du clavier, il effectue une opération d'**entrée** (in) ou de lecture de données.

Dans le langage Java, les opérations de sortie sont réalisées grâce à l'instruction `System.out.print()`, qui permet d'afficher des informations à l'écran. Par exemple, l'instruction :

```
System.out.print(F + " francs valent " + E + " euros") ;
```

affiche à l'écran le contenu de la variable `F`, suivi du texte `francs valent`, puis le contenu de la variable `E`, suivi du texte `euros`.

Pour distinguer le commentaire du nom de variable, le commentaire est placé entre guillemets. Le contenu de la variable est affiché en réunissant la variable au commentaire à l'aide du signe `+`.

Pour afficher des résultats sur plusieurs lignes, il convient d'utiliser l'instruction :

```
System.out.println()
```

Avec la version Java 1.5, les opérations d'entrée sont réalisées par l'intermédiaire de la classe `Scanner` grâce aux instructions suivantes :

```
Scanner lectureClavier = new Scanner(System.in);
int i = lectureClavier.nextInt();
```

Les méthodes de lecture ont pour nom d'appel :

- `nextByte()` pour saisir une valeur de type `byte` ;
- `nextShort()` pour saisir une valeur de type `short` ;
- `nextInt()` pour saisir une valeur de type `int` ;
- `nextLong()` pour saisir une valeur de type `long` ;
- `nextFloat()` pour saisir une valeur de type `float` ;
- `nextDouble()` pour saisir une valeur de type `double` ;
- `next()` pour saisir une valeur de type `String` ;
- `next().charAt(0)` pour saisir une valeur de type `char`.

Copyright © 2016 Eyrolles.

Exercices

Comprendre les opérations de sortie

Exercice 2.1 Soit un programme Java contenant les déclarations

```
int i = 223, j = 135 ;
float a = 335.5f, b = 20.5f ;
char R = 'R', T = 'T' ;
```

Décrivez l'affichage généré par chacune des instructions suivantes :

```
System.out.println("Vous avez entre : " + i) ;
System.out.println("Pour un montant de "+ a + " le total vaut : "+ i + j);
System.out.print("Apres reduction de " + b + " %, vous gagnez : ") ;
System.out.println( (a*b)/100 + " euros") ;
System.out.print(" La variable R = " + R + " et T = " + T) ;
```

Exercice 2.2 En tenant compte des déclarations de variables suivantes, écrivez les instructions `System.out.print()` de façon à obtenir l'affichage suivant :

double x = 4, y = 2 ;	double x = 9, y = 3 ;
`x = 4.0` et `y = 2.0`	`x = 9.0` et `y = 3.0`
Racine carrée de 4.0 = 2.0	Racine carrée de 9.0 = 3.0
4.0 a la puissance 2.0 = 16.0	9.0 a la puissance 3.0 = 729.0

Remarque Notez que la racine carrée de `x` s'obtient par la fonction `Math.sqrt(x)` et que a^b se calcule avec la méthode `Math.pow(a,b)` .

Comprendre les opérations d'entrée

Exercice 2.3 Pour chacun des deux programmes suivants, et compte tenu des informations fournies par l'utilisateur, quelles sont les valeurs affichées à l'écran ?

Copyright © 2016 Eyrolles.

L'utilisateur fournit au clavier 2, puis 3, puis 4	L'utilisateur fournit au clavier 2

`int X, Y ;`
`X = lectureClavier.nextInt();`
`Y = lectureClavier.nextInt();`
`X = lectureClavier.nextInt();`
`X = X+Y ;`
`System.out.print(" X = " + X) ;`
`System.out.print(" Y = " + Y) ;`
	`int X, Y ;`
	`X = lectureClavier.nextInt();`
	`Y = 0 ;`
	`X = X+Y ;`
	`System.out.println(" X = " + X) ;`
	`System.out.println(" Y = " + Y) ;`

Observer et comprendre la structure d'un programme Java

Exercice 2.4 En prenant exemple sur la structure suivante, écrivez un programme `Euro` qui convertit des francs en euros. (Rappel : 1 euro = 6,55957 francs) :

```
public class .....// Donner un nom à la classe
{
  public static void main(String [] argument)
  {
   // Déclarer les variables représentant les francs et les euros
   // ainsi que le taux de conversion
   .....
   // Déclaration de la variable représentant la lecture au clavier
   .....
   // Afficher et saisir le nombre de francs
   .....
   .....
   // Calculer le nombre d'euros
   .....
   // Afficher le résultat suivant l'exemple donné ci-dessous
   .....
  }
}
```

L'affichage du résultat se fera sous la forme suivante :

```
Nombre de francs :  120
Conversion F/E   :  6,55957
Nombre d'euros   :  18,293
```

Copyright © 2016 Eyrolles.

Le projet : Gestion d'un compte bancaire

Afficher le menu principal ainsi que ses options

L'objectif de ce premier programme est d'écrire toutes les instructions qui permettent l'affichage des menus définis dans le cahier des charges décrit au chapitre introductif, « Naissance d'un programme », ainsi que la saisie des données demandées. Le programme affiche tous les messages de toutes les options, sans contrôle sur le choix de l'utilisateur.

- Le menu principal s'affiche de la façon suivante :

```
1. Creer un compte
2. Afficher un compte
3. Creer une ligne comptable
4. Sortir
5. De l'aide
Votre choix :
```

- Une fois le menu affiché, le programme attend la saisie du choix de l'utilisateur.
- L'option `1` du menu principal a pour affichage :

```
Type du compte [Types possibles : courant, joint, épargne] :
Numero du compte :
Première valeur creditee :
Taux de placement :
```

  L'option `2` réalise les opérations suivantes :

  – Affiche la demande de saisie du numéro du compte que l'utilisateur souhaite consulter.

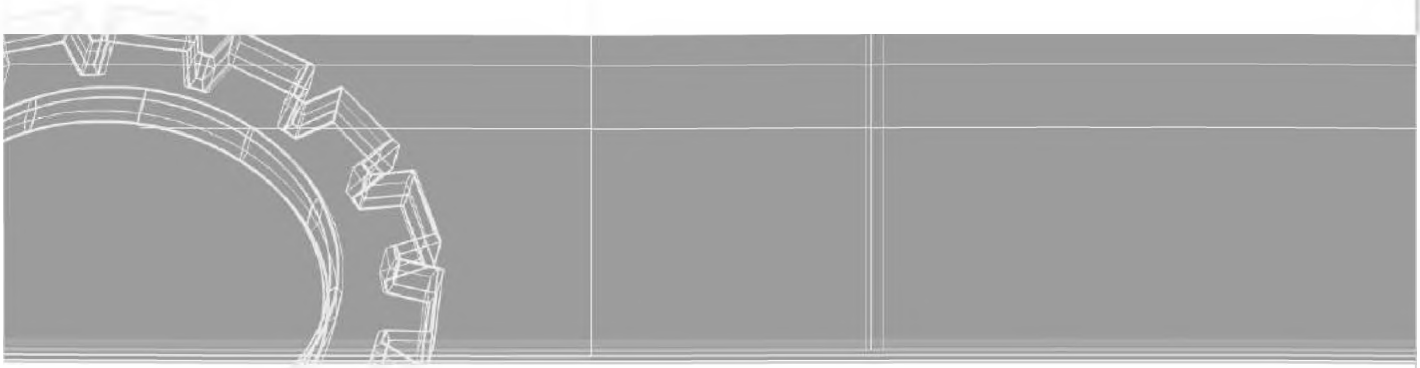
  – Saisit le numéro de compte.

- L'option `3` affiche : `" option non programmee "`.
- L'option `4` termine l'exécution du programme. Pour cela, utilisez la fonction Java `System.exit(0) ;`.
- Avec l'option `5`, le programme affiche une ligne d'explication pour chaque option du menu principal.

Copyright © 2016 Eyrolles.

Chapitre 3

Faire des choix



Une fois les variables définies et les valeurs stockées en mémoire, l'ordinateur est capable de les **tester** ou de les **comparer** de façon à réaliser une instruction plutôt qu'une autre, suivant le résultat de la comparaison.

Le programme n'est alors plus exécuté de façon séquentielle (de la première ligne jusqu'à la dernière). L'ordre est rompu, une ou plusieurs instructions étant ignorées en fonction du résultat du test. Le programme peut s'exécuter, en tenant compte de contraintes imposées par le programmeur.

Dans ce chapitre, nous abordons la notion de choix ou de test, en reprenant l'algorithme du café chaud, pour le transformer en un algorithme du café chaud sucré **ou** non (voir section « L'algorithme du café chaud, sucré ou non »).

Ensuite, à la section « L'instruction if-else », nous étudions la structure `if-else` proposée par le langage Java, qui permet de réaliser des choix.

Enfin, à la section « L'instruction switch, ou comment faire des choix multiples », nous examinons le concept de choix multiples par l'intermédiaire de la structure `switch`.

L'algorithme du café chaud, sucré ou non

Pour mieux comprendre la notion de choix, nous allons reprendre l'algorithme du café chaud pour le transformer en `algorithme du café chaud, sucré ou non`. L'énoncé ainsi transformé nous oblige à modifier la liste des objets manipulés, ainsi que celle des opérations à réaliser.

Copyright © 2016 Eyrolles.

Définition des objets manipulés

Pour obtenir du café sucré, nous devons ajouter à notre liste un nouvel ingrédient, le sucre, et un nouvel ustensile, la petite cuillère.

```
café moulu
filtre
eau
cafetière électrique
tasse
électricité
table
sucre
petite cuillère
```

Liste des opérations

De la même façon, nous devons modifier la liste des opérations, de façon qu'elle prenne en compte les nouvelles données :

```
Verser l'eau dans la cafetière, le café dans la tasse, le café dans
le filtre.
Prendre du café moulu, une tasse, de l'eau, une cafetière électrique,
un filtre, un morceau de sucre, une petite cuillère.
Brancher, allumer ou éteindre la cafetière électrique.
Attendre que le café soit prêt.
Poser la tasse, la cafetière sur la table, le filtre dans la
cafetière, le sucre dans la tasse, la petite cuillère dans la tasse.
```

Ordonner la liste des opérations

Ainsi modifiée, la liste des opérations doit être réordonnée afin de rechercher le moment le mieux adapté pour ajouter les nouvelles opérations :

- En décidant de prendre le sucre et la petite cuillère en même temps que le café et le filtre, nous plaçons les nouvelles instructions `"prendre..."` entre les instructions 2 et 3 définies à la section « Ordonner la liste des opérations » du chapitre introductif, « Naissance d'un programme ».
- En décidant de poser le sucre et la petite cuillère dans la tasse avant d'y verser le café, nous écrivons les nouvelles instructions `" poser..."` avant l'instruction 15 du même exemple.

Copyright © 2016 Eyrolles.

Nous obtenons la liste des opérations suivantes :

```
 0. Prendre une cafetière.
 1. Poser la cafetière sur la table.
 2. Prendre du café.
 3. Prendre un morceau de sucre.
 4. Prendre une petite cuillère.
 5. Prendre un filtre.
 6. Verser le café dans le filtre.
 7. Prendre de l'eau.
 8. Verser l'eau dans la cafetière.
 9. Brancher la cafetière.
10. Allumer la cafetière.
11. Attendre que le café soit prêt.
12. Prendre une tasse.
13. Poser la tasse sur la table.
14. Poser le sucre dans la tasse.
15. Poser la petite cuillère dans la tasse.
16. Éteindre la cafetière.
17. Verser le café dans la tasse.
```

Écrite ainsi, cette marche à suivre nous permet d'obtenir un café chaud sucré. Elle ne nous autorise pas à choisir entre sucré ou non. Pour cela, nous devons introduire un test, en posant une condition devant chaque instruction concernant la prise du sucre, c'est-à-dire :

```
 0. Prendre une cafetière.
 1. Poser la cafetière sur la table.
 2. Prendre du café.
 3. Si (café sucré) Prendre un morceau de sucre.
 4. Si (café sucré) Prendre une petite cuillère.
 5. Prendre un filtre.
 6. Verser le café dans le filtre.
 7. Prendre de l'eau.
 8. Verser l'eau dans la cafetière.
 9. Brancher la cafetière.
10. Allumer la cafetière.
11. Attendre que le café soit prêt.
12. Prendre une tasse.
13. Poser la tasse sur la table.
14. Si (café sucré) Poser le sucre dans la tasse.
15. Si (café sucré) Poser la petite cuillère dans la tasse.
16. Éteindre la cafetière.
17. Verser le café dans la tasse.
```

Copyright © 2016 Eyrolles.

Dans cette situation, nous obtenons du café sucré ou non, selon notre choix. Observons cependant que le test `Si (café sucré)` est identique pour les instructions `3`, `4`, `14` et `15`. Pour cette raison, et sachant que chaque test représente un coût en termes de temps d'exécution, il est conseillé de regrouper au même endroit toutes les instructions relatives à un même test.

C'est pourquoi nous distinguons deux blocs d'instructions distincts :

- les instructions soumises à la condition de café sucré (*II Préparer le sucre*) ;
- les instructions réalisables quelle que soit la condition (*I Préparer le café*).

Dans ce cas, la nouvelle solution s'écrit :

Instructions	Bloc d'instructions
0. Prendre une cafetière. 1. Poser la cafetière sur la table. 2. Prendre du café. 3. Prendre un filtre. 4. Verser le café dans le filtre. 5. Prendre de l'eau. 6. Verser l'eau dans la cafetière. 7. Brancher la cafetière. 8. Allumer la cafetière. 9. Attendre que le café soit prêt. 10. Prendre une tasse. 11. Poser la tasse sur la table. 12. Éteindre la cafetière. 13. Verser le café dans la tasse.	*I Préparer le café*
Si (café sucré)	
1. Prendre un morceau de sucre. 2. Prendre une petite cuillère. 3. Poser le sucre dans la tasse. 4. Poser la petite cuillère dans la tasse.	*II Préparer le sucre*

La réalisation du bloc *I Préparer le café* nous permet d'obtenir du café chaud. Ensuite, en exécutant le test `Si (café sucré)`, deux solutions sont possibles :

- La proposition `(café sucré)` est vraie, et alors les instructions 1 à 4 du bloc *II Préparer le sucre* sont exécutées. Nous obtenons du café chaud sucré.
- La proposition `(café sucré)` est fausse, et les instructions qui suivent ne sont pas exécutées. Nous obtenons un café non sucré.

Pour programmer un choix, nous avons écrit une condition devant les instructions concernées. En programmation, il en est de même. Le langage Java propose plusieurs instructions de test,

Copyright © 2016 Eyrolles.

à savoir la structure `if-else`, que nous étudions ci-après, et la structure `switch` que nous analysons à la section « L'instruction switch, ou comment faire des choix multiples », un peu plus loin dans ce chapitre.

L'instruction if-else

L'instruction `if-else` se traduit en français par les termes `si-sinon`. Elle permet de programmer un choix, en plaçant derrière le terme `if` une condition, comme nous avons placé une condition derrière le terme `si` de l'algorithme du café chaud, sucré ou non.

L'instruction `if-else` se construit de la façon suivante :

- en suivant une *syntaxe*, ou forme, précise du langage Java (voir « Syntaxe d'if-else ») ;
- en précisant la condition à tester (voir « Comment écrire une condition »).

Nous présentons en fin de cette section un exemple de programme qui recherche la plus grande des deux valeurs saisies au clavier (voir « Rechercher le plus grand de deux éléments »).

Syntaxe d'if-else

L'écriture de l'instruction `if-else` obéit aux règles de syntaxe suivantes :

```
if   (condition)    // si la condition est vraie
{                   // faire
  plusieurs instructions ;
}                   // fait
else                // sinon (la condition ci-dessus est fausse)
{                   //faire
  plusieurs instructions ;
}                   //fait
```

- Si la condition située après le mot-clé `if` et placée obligatoirement entre parenthèses est vraie, alors les instructions placées dans le bloc défini par les accolades ouvrante et fermante immédiatement après sont exécutées.
- Si la condition est fausse, alors les instructions définies dans le bloc situé après le mot-clé `else` sont exécutées.

De cette façon, un seul des deux blocs peut être exécuté à la fois, selon que la condition est vérifiée ou non.

Signalons que :

- La ligne d'instruction `if (condition)` ou `else` ne se termine jamais par un point-virgule (`;`).

Copyright © 2016 Eyrolles.

- Les accolades { et } définissent un bloc d'instructions. Cela permet de regrouper ensemble toutes les instructions relatives à un même test.
- L'écriture du bloc `else` n'est pas obligatoire. Il est possible de n'écrire qu'un bloc `if` sans programmer d'instruction dans le cas où la condition n'est pas vérifiée (comme dans l'algorithme du café chaud, sucré ou non). En d'autres termes, il peut y avoir des `if` sans `else`.
- S'il existe un bloc `else`, celui-ci est obligatoirement « accroché » à un `if`. Autrement dit, il ne peut y avoir d'`else` sans `if`.
- Le langage Java propose une syntaxe simplifiée lorsqu'il n'y a qu'une seule instruction à exécuter dans l'un des deux blocs `if` ou `else`. Dans ce cas, les accolades ouvrante et fermante ne sont pas obligatoires :

```
if   (condition)   une seule instruction ;
else    une seule instruction ;
```

ou :

```
if   (condition)
{               // faire
  plusieurs instructions ;
}               // fait
else    une seule instruction ;
```

ou encore :

```
if   (condition)   une seule instruction ;
else
{               // faire
  plusieurs instructions ;
}               // fait
```

Une fois connue la syntaxe générale de la structure `if-else`, nous devons écrire la condition (placée entre parenthèses, juste après `if`) permettant à l'ordinateur d'exécuter le test.

Comment écrire une condition

L'écriture d'une condition en Java fait appel aux notions d'opérateurs relationnels et conditionnels.

Les opérateurs relationnels

Une condition est formée par l'écriture de la comparaison de deux expressions, une expression pouvant être une valeur numérique ou une expression arithmétique. Pour comparer deux

Copyright © 2016 Eyrolles.

expressions, le langage Java dispose de six symboles représentant les opérateurs relationnels traditionnels en mathématiques.

Opérateur	Signification pour des valeurs numériques	Signification pour des valeurs de type caractère
`==`	égal	identique
`<`	inférieur strictement	plus petit dans l'ordre alphabétique
`<=`	inférieur ou égal	plus petit ou identique dans l'ordre alphabétique
`>`	supérieur strictement	plus grand dans l'ordre alphabétique
`>=`	supérieur ou égal	plus grand ou identique dans l'ordre alphabétique
`!=`	différent	différent

Un opérateur relationnel permet de comparer deux expressions de même type. Il n'est pas possible de comparer un réel avec un entier ou un entier avec un caractère.

Lorsqu'il s'agit de comparer deux expressions composées d'opérateurs arithmétiques (`+ - * / %`), les opérateurs relationnels sont moins prioritaires par rapport aux opérateurs arithmétiques. De cette façon, les expressions mathématiques sont d'abord calculées avant d'être comparées.

Notons que pour tester l'égalité entre deux expressions, nous devons utiliser le symbole == et non pas un simple =. En effet, en Java, le signe = n'est pas un signe d'égalité au sens de la comparaison mais le signe de l'affectation, qui permet de placer une valeur dans une variable.

Question

En initialisant les variables *a*, *b*, *lettre* et *car* de la façon suivante :

```
int a = 3, b = 5 ;
char lettre = 'i', car = 'j' ;
```

examinez si les conditions suivantes sont vraies ou fausses :

```
(a != b)
(a + 2 == b)
(a + 8 < 2 * b)
(lettre <= car)
(lettre == 'w')
```

Réponse

La condition `(a != b)` est vraie car 3 est différent de 5.

La condition `(a + 2 == b)` est vraie car 3 + 2 vaut 5.

La condition `(a + 8 < 2 * b)` est fausse car 3 + 8 est plus grand que 2 * 5.

La condition `(lettre <= car)` est vraie car le caractère `'i'` est placé avant `'j'` dans l'ordre alphabétique.

La condition `(lettre == 'w')` est fausse car le caractère `'i'` est différent du caractère `'w'`.

Copyright © 2016 Eyrolles.

Les opérateurs logiques

Les opérateurs logiques sont utilisés pour associer plusieurs conditions simples et, de cette façon, créer des conditions multiples en un seul test. Il existe trois grands opérateurs logiques, symbolisés par les caractères suivants :

Opérateur	Signification
!	NON logique
&&	ET logique
||	OU logique

Question En initialisant les variables `x`, `y`, `z` et `r` de la façon suivante :

```
int x = 3, y = 5, z = 2, r = 6 ;
```

examinez si les conditions suivantes sont vraies ou fausses :

```
(x < y) && (z < r)
(x > y) || (z < r)
!(z < r)
```

Réponse Sachant que la condition `(x < y) && (z < r)` est vraie si les deux expressions `(x < y)` **et** `(z < r)` sont toutes les deux vraies et devient fausse si l'une des deux expressions est fausse, l'expression donnée en exemple est vraie. En effet `(3 < 5)` est vraie et `(2 < 6)` est vraie.

Sachant que la condition `(x > y) || (z < r)` est vraie si l'une des expressions `(x > y)` **ou** `(z < r)` est vraie et devient fausse si les deux expressions sont fausses, l'expression donnée en exemple est vraie car `(3 > 5)` est fausse, mais `(2 < 6)` est vraie.

Sachant que la condition `!(z < r)` est vraie si l'expression `(z < r)` est fausse et devient fausse si l'expression est vraie, alors l'expression donnée en exemple est fausse car `(2 < 6)` est vraie.

Rechercher le plus grand de deux éléments

Pour mettre en pratique les notions théoriques abordées aux deux sections précédentes, nous allons écrire un programme qui affiche, dans l'ordre croissant, deux valeurs entières saisies au clavier et recherche la plus grande des deux. Pour cela, nous devons :

1. Demander la saisie de deux valeurs au clavier.
2. Tester si la première valeur saisie est plus grande que la seconde.
   a. Si tel est le cas :
      - afficher dans l'ordre croissant, en affichant la seconde valeur saisie puis la première ;
      - stocker la plus grande des valeurs dans une variable spécifique, soit la première valeur.
   b. Sinon :
      - afficher dans l'ordre croissant, en affichant la première valeur saisie puis la seconde ;
      - stocker la plus grande des valeurs dans une variable spécifique, soit la seconde valeur.

Copyright © 2016 Eyrolles.

3. Afficher la plus grande des valeurs.

Nous devons, dans un premier temps, déclarer trois variables entières, deux pour les valeurs à saisir et une pour stocker la plus grande des deux. Nous écrivons l'instruction de déclaration suivante :

```
int première, deuxième, laPlusGrande ;
```

1. La saisie des deux valeurs est ensuite réalisée par (voir le chapitre 2, « Communiquer une information ») :

```
Scanner lectureClavier = new Scanner(System.in);
System.out.print("Entrer une valeur :") ;
première = lectureClavier.nextInt();
System.out.print("Entrer une deuxieme valeur :") ;
deuxième = lectureClavier.nextInt();
```

2. Pour tester si la première valeur saisie est plus grande que la seconde, l'instruction `if` s'écrit :

```
if (première > deuxième)
```

   a. Deux instructions composent ce test : l'affichage dans l'ordre croissant puis le stockage de la plus grande valeur. Il est donc nécessaire de les placer dans un bloc défini par une accolade ouvrante ({) et une accolade fermante (}) :

```
{
  // Afficher les valeurs dans l'ordre croissant
  System.out.println(deuxième + "  " + première) ;
  // Stocker la plus grande dans une variable spécifique
  laPlusGrande = première ;
}
```

   b. De la même façon, le cas contraire est décrit par l'instruction `else` et est composé de deux instructions. Nous avons donc :

```
else
{
  // Afficher les valeurs dans l'ordre croissant
  System.out.println(première + " " + deuxième) ;
  // Stocker la plus grande dans une variable spécifique
  laPlusGrande = deuxième ;
}
```

3. Nous affichons enfin la plus grande valeur par l'instruction :

```
System.out.println("La plus grande valeur est : " + laPlusGrande) ;
```

Ce message est affiché dans tous les cas, et l'instruction est donc placée en dehors de toute structure conditionnelle.

Copyright © 2016 Eyrolles.

Pour finir, le programme est placé dans une fonction `main()` et une classe, que nous appelons `Maximum`, puisqu'il s'agit ici de trouver la valeur maximale de deux valeurs. De cette façon, le programme peut être compilé et exécuté.

Exemple : code source complet

```
import java.util.*;
public class Maximum  // Le fichier s'appelle Maximum.java
{
  public static void main (String [] parametre)
  {
   int première, deuxième, laPlusGrande ;
   Scanner lectureClavier = new Scanner(System.in);
   System.out.print("Entrer une valeur :") ;
   première = lectureClavier.nextInt();
   System.out.print("Entrer une deuxieme valeur :") ;
   deuxième = lectureClavier.nextInt();
   if (première > deuxième)
   {
     System.out.println(deuxième + "  " + première) ;
     laPlusGrande = première ;
   }
   else
   {
     System.out.println(première + " " + deuxième) ;
     laPlusGrande = deuxième ;
   }
   System.out.println("La plus grande valeur est : " +
                      laPlusGrande) ;
  } // Fin du main ()
} // Fin de la Class Maximum
```

Question Que se passe-t-il si l'utilisateur entre les valeurs suivantes :

(Les caractères grisés sont des valeurs choisies par l'utilisateur.)

```
Entrer une valeur : 3
Entrer une deuxieme valeur : 5
```

Réponse Le programme affiche la réponse suivante :

```
3  5
La plus grande valeur est : 5
```

La première valeur étant plus petite que la seconde, le programme n'exécute que les instructions placées dans le bloc `else`.

Copyright © 2016 Eyrolles.

Deux erreurs à éviter

Deux types d'erreurs sont à éviter par le programmeur débutant. Il s'agit des erreurs issues d'une mauvaise construction des blocs `if` ou `else` et d'un placement incorrect du point-virgule.

La construction de blocs

Reprenons l'exemple précédent en l'écrivant comme suit :

```
if (première > deuxième)
     System.out.println(deuxième + "  " + première) ;
     laPlusGrande = première ;
else
   {
     System.out.println(première+" "+deuxième) ;
     laPlusGrande = deuxième ;
   }
```

En exécutant pas à pas cet extrait de programme, nous observons qu'il n'y a pas d'accolade ouvrante ({) derrière l'instruction `if`. Cette dernière ne possède donc pas de bloc composé de plusieurs instructions. Seule l'instruction d'affichage `System.out.println(deuxième + "  " + première) ;` se situe dans `if`. L'exécution d'`if` s'achève donc juste après l'affichage des valeurs dans l'ordre croissant.

Ensuite, l'instruction `laPlusGrande = première ;` est théoriquement exécutée en dehors de toute condition. Cependant, l'instruction suivante est `else`, alors que l'instruction `if` s'est achevée précédemment. Le compilateur ne peut attribuer ce `else` à un `if`. Il y a donc erreur de compilation du type `'else' without 'if'`.

De la même façon, il y a erreur de compilation lorsque le programme est construit sur la forme suivante :

```
if (première > deuxième)
{....
}
laPlusGrande = première ;
else
{...
}
```

Le point-virgule

Dans le langage Java, le point-virgule constitue une instruction à part entière, qui représente l'instruction vide. Par conséquent, écrire le programme suivant ne provoque aucune erreur à la compilation :

```
if (première > deuxième) ;
     System.out.println(deuxième + "  " + première) ;
```

Copyright © 2016 Eyrolles.

L'exécution de cet extrait de programme a pour résultat :

Si `première` est plus grand que `deuxième`, l'ordinateur exécute le ; (point-virgule) situé immédiatement après la condition, c'est-à-dire rien. L'instruction `if` est terminée, puisqu'il n'y a pas d'accolades ouvrante et fermante. Seule l'instruction ; est soumise à `if`.

Le message affichant les valeurs par ordre croissant ne fait pas partie du test. Il est donc affiché, quelles que soient les valeurs de `première` et `deuxième`.

Des if-else imbriqués

Dans le cas de choix arborescents – un choix étant fait, d'autres choix sont à faire, et ainsi de suite –, il est possible de placer des structures `if-else` à l'intérieur d'`if-else`. On dit alors que les structures `if-else` sont imbriquées les unes dans les autres.

Lorsque ces imbrications sont nombreuses, il est possible de les représenter à l'aide d'un graphique de structure arborescente, dont voici un exemple :

Imbrications d'`if else`

```
if (Condition 1)
{
  if (Condition 2)
  {
   instruction A
  }
  else
  {
   instruction B
  }
}
else
{
  instruction C
}
```

Représentation du choix arborescent

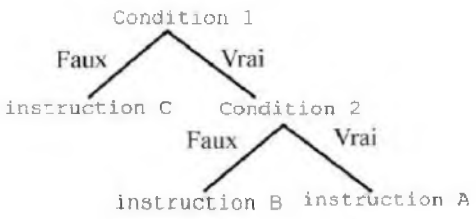


Figure 3-1

Quand il y a moins d'else que d'if

Une instruction `if` peut ne pas contenir d'instruction `else`. Dans de tels cas, il peut paraître difficile de savoir à quel `if` est associé le dernier `else`.

Copyright © 2016 Eyrolles.

Comparons les deux exemples suivants :

Imbrications d'`if else` — **Arbre des choix**

```
if (Condition 1)
{
  if (Condition 2)
  {
    if (Condition 3)
    {
      instruction A
    }
    else
    {
      instruction B
    }
  }
}
else
{
  instruction C
}
```

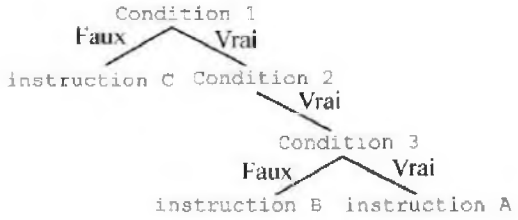


Figure 3-2

```
if (Condition 1)
{
  if (Condition 2)
  {
    if (Condition 3)
    {
      instruction A
    }
    else
    {
      instruction B
    }
  }
  else
  {
    instruction C
  }
}
```

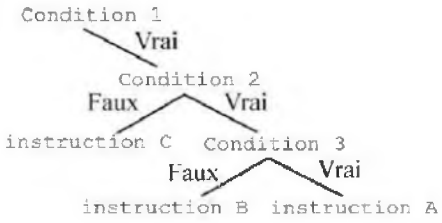


Figure 3-3

Du premier au deuxième exemple, par le jeu des fermetures d'accolades, le dernier bloc `else` est déplacé d'un bloc vers le haut. Ce déplacement modifie la structure arborescente. Les algorithmes associés ont des résultats totalement différents.

Copyright © 2016 Eyrolles.

Remarques Pour déterminer une relation `if-else`, observons qu'un « bloc `else` » se rapporte toujours au dernier « bloc `if` » rencontré, auquel un `else` n'a pas encore été attribué.

Les blocs `if` et `else` étant délimités par les accolades ouvrantes et fermantes, il est conseillé, pour éviter toute erreur, de bien relier chaque parenthèse ouvrante avec sa fermante.

L'instruction switch, ou comment faire des choix multiples

Lorsque le nombre de choix possible est plus grand que deux, l'utilisation de la structure `if-else` devient rapidement fastidieuse. Les imbrications des blocs demandent à être vérifiées avec précision, sous peine d'erreur de compilation ou d'exécution.

C'est pourquoi, le langage Java propose l'instruction `switch` (traduire par selon, ou suivant), qui permet de programmer des choix multiples selon une syntaxe plus claire.

Construction du switch

L'écriture de l'instruction `switch` obéit aux règles de syntaxe suivantes :

```
switch (valeur)
{
  case étiquette 1 :
      // Une ou plusieurs instructions
  break ;
  case étiquette 2 :
  case étiquette 3 :
      // Une ou plusieurs instructions
  break ;
  default :
      // Une ou plusieurs instructions
}
```

La variable `valeur` est évaluée. Suivant cette valeur, le programme recherche l'`étiquette` correspondant à la valeur obtenue et définie à partir des instructions `case` *`étiquette`*.

- Si le programme trouve une étiquette correspondant au contenu de la variable `valeur`, il exécute la ou les instructions qui suivent l'étiquette, jusqu'à rencontrer le mot-clé `break`.
- S'il n'existe pas d'étiquette correspondant à `valeur`, alors le programme exécute les instructions de l'étiquette `default`.

Copyright © 2016 Eyrolles.

- Le type de la variable `valeur` ne peut être que `char` ou `int`, `byte`, `short` ou `long`. Il n'est donc pas possible de tester des valeurs réelles.
- Une étiquette peut contenir aucune, une ou plusieurs instructions.
- L'instruction `break` permet de sortir du bloc `switch`. S'il n'y a pas de `break` pour une étiquette donnée, le programme exécute les instructions de l'étiquette suivante.

Remarques Notes sur la version Java 7

À partir de la version 7 de Java, le test sur des chaînes de caractères est autorisé. Il devient possible de réaliser un `switch` en utilisant des mots comme étiquette. Par exemple :

```
String choix ;
Scanner lectureClavier = new Scanner(System.in);
System.out.println("Votre choix  (oui/non) ? : ") ;
choix = lectureClavier.nextLine();

switch (choix)
{
  case  "oui" :
        System.out.println("Vous avez saisi oui !") ;
  break ;
  case "non" :
       System.out.println("Vous avez saisi non !") ;
  break ;
  default :
       System.out.println("Vous avez saisi ni oui ni non !") ;
}
```

Calculer le nombre de jours d'un mois donné

Pour mettre en pratique les notions théoriques abordées à la section précédente, nous allons écrire un programme qui calcule et affiche le nombre de jours d'un mois donné.

Le nombre de jours dans un mois peut varier entre les valeurs `28`, `29`, `30` ou `31`, suivant le mois et l'année. Les mois de janvier, mars, mai, juillet, août, octobre et décembre sont des mois de `31` jours. Les mois d'avril, juin, septembre et novembre sont des mois de `30` jours. Seul le mois de février est particulier, puisque son nombre de jours est de `29` jours pour les années bissextiles et de `28` jours dans le cas contraire. Sachant cela, nous devons :

- Demander la saisie au clavier du numéro du mois ainsi que de l'année recherchée.

Copyright © 2016 Eyrolles.

- Créer autant d'étiquettes qu'il y a de mois dans une année, c'est-à-dire `12`. Compte tenu du fonctionnement de la structure `switch`, chaque étiquette est une valeur entière correspondant au numéro du mois de l'année (`1` pour janvier, `2` pour février, etc.).
- Regrouper les étiquettes relatives aux mois à `31` jours et stocker cette dernière valeur dans une variable spécifique.
- Regrouper les étiquettes relatives aux mois à `30` jours et stocker cette dernière valeur dans une variable spécifique.
- Pour l'étiquette relative au mois de février, tester la valeur de l'année pour savoir si l'année concernée est bissextile ou non. Une année est bissextile tous les `quatre` ans, sauf lorsque le millésime est divisible par `100` et non pas par `400`. En d'autres termes, pour qu'une année soit bissextile, il suffit que l'année soit un nombre divisible par 4 et non divisible par `100` ou alors par `400`. Dans tous les autres cas, l'année n'est pas bissextile.

Compte tenu de toutes ces remarques, nous devons dans un premier temps déclarer trois variables entières, une pour représenter le mois, la deuxième l'année, et la troisième le nombre de jours par mois. Sachant que le mois et le nombre de jours par mois ne dépassent jamais la valeur `127`, nous pouvons les déclarer de type `byte`. Pour l'année, le type `short` suffit, puisque les valeurs de ce type peuvent aller jusqu'à `32767`.

Exemple : code source complet

```
import java.util.*;
public class JourParMois  // Le fichier s'appelle JourParMois.java
{
  public static void main (String [] parametre)
  {
   byte mois, nbjours = 0 ;
   short année ;
   Scanner lectureClavier = new Scanner(System.in);
   System.out.println("De quel mois s'agit-il ? : ") ;
   mois = lectureClavier.nextByte();
   System.out.println("De quelle annee ? : ") ;
   année = lectureClavier.nextShort();
   switch(mois)
   {
     case 1 : case 3 :    // Pour les mois à 31 jours
     case 5 : case 7 :
     case 8 : case 10 :
     case 12 :
            nbjours = 31 ;
            break ;
     case 4 : case 6 :    // Pour les mois à 30 jours
```

Copyright © 2016 Eyrolles.

```
        case 9 : case 11 :
                nbjours  = 30 ;
                break ;
        case 2 :          // Pour le cas particulier du mois de février
              if (année % 4 == 0 && année % 100 != 0 || année %
                                400 == 0)
                  nbjours  = 29 ;
              else  nbjours  = 28 ;
              break ;
        default :         // En cas d'erreur de frappe
             System.out.println("Impossible, ce mois n'existe pas ") ;
              System.exit(0) ;
      }
      System.out.print(" En " + annee + ", le mois n° " + mois) ;
      System.out.println(" a " + nbjours + " jours ") ;
    } // Fin du main()
} // Fin de la class JourParMois
```

Question Que se passe-t-il si l'utilisateur entre les valeurs suivantes :

```
De quel mois s'agit-il ? : 5
De quelle annee ? : 1999
```

Réponse Le programme affiche la réponse suivante :

```
En 1999 le mois n° 5 a 31 jours
```

Le programme recherche l'étiquette 5. Il exécute les instructions qui suivent jusqu'à rencontrer un `break`. Pour l'étiquette 5, le programme exécute les instructions des étiquettes 7, 8, 10 et 12 car ces étiquettes ne possèdent ni instruction, ni `break`. Seule l'étiquette 12 possède une instruction, qui affecte la valeur 31 à la variable `nbjours`. L'instruction `break` qui suit permet de sortir de la structure `switch`. Le programme exécute enfin l'instruction située immédiatement après le `switch`, c'est-à-dire l'affichage du message annonçant le résultat.

Question Que se passe-t-il si l'utilisateur entre les valeurs suivantes :

```
De quel mois s'agit-il ? : 2
De quelle annee ? : 2000
```

Réponse Le programme affiche la réponse suivante :

```
En 2000 le mois n° 2 a 29 jours
```

Ici, le programme va directement à l'étiquette 2, qui est composée d'un test sur l'année pour savoir si l'année est bissextile. Une année est bissextile lorsque son millésime est divisible par 4, à l'exception des années dont le millésime est divisible par 100 et non pas par 400. La valeur 2 000 est divisible par 4, 100 et 400 puisque le reste de la division entière (%) de 2000 par 4, 100 ou 400 est nul. La variable `nbjours` prend donc la valeur 29. Le programme sort ensuite du `switch` grâce à l'instruction `break` qui suit et exécute pour finir l'affichage du résultat.

Copyright © 2016 Eyrolles.

Question Que se passe-t-il si l'utilisateur entre les valeurs suivantes :

```
De quel mois s'agit-il ? : 15
De quelle annee ? : 1999
```

Réponse Le programme affiche la réponse suivante :

```
Impossible, ce mois n'existe pas
```

L'étiquette `15` n'étant pas définie dans le bloc `switch`, le programme exécute les instructions qui composent l'étiquette `default`. Le programme affiche un message d'erreur et termine son exécution grâce à l'instruction `System.exit(0) ;`

Remarque Grâce à l'étiquette `default`, le programme connaît les instructions à exécuter dans le cas de choix « anormaux » (erreur de frappe, par exemple, ou valeur saisie n'entrant pas dans l'intervalle des valeurs possibles traitées par le programme). De cette façon, il devient possible de prévenir d'éventuelles erreurs pouvant causer l'arrêt brutal de l'exécution du programme.

Comment choisir entre if-else et switch ?

La structure `switch` ne permet de tester que des égalités de valeurs entières (`byte`, `short`, `int` ou `long`) ou de type caractère `char` (et `String` à partir de la version 7 de Java). Elle **ne peut** donc **pas** être utilisée pour :

- Tester des valeurs réelles (`float` ou `double`).
- Rechercher si la valeur est plus grande, plus petite ou différente d'une certaine étiquette.

Par contre, l'instruction `if-else` peut être employée dans tous les cas en testant tout type de variable, selon toute condition.

Remarques Si une condition parmi d'autres conditions envisagées a une plus grande probabilité d'être satisfaite, celle-ci doit être placée en premier test dans une structure `if-else`, de façon à éviter à l'ordinateur d'effectuer de trop nombreux tests inutiles.

Si toutes les conditions ont une probabilité voisine ou équivalente d'être réalisées, la structure `switch` est plus efficace. Elle ne demande qu'une seule évaluation, alors que dans les instructions `if-else` imbriquées, chaque condition doit être évaluée.

Copyright © 2016 Eyrolles.

Résumé

L'instruction `if-else` (traduction : si, sinon) permet de programmer des choix. De façon générale, l'instruction `if-else` s'écrit :

```
if   (condition)
// si la condition est vraie
{     // faire
  plusieurs instructions ;
}     // fait
else  // sinon
{     //faire
  plusieurs instructions ;
}     //fait
```

Ou encore

```
if (condition) une seule
instruction ;
else     une seule instruction ;
```

- Si la condition située après le mot-clé `if` (placée obligatoirement entre parenthèses) est vraie, alors les instructions placées dans le bloc défini par les accolades ouvrante et fermante immédiatement après sont exécutées.
- Si la condition est fausse, alors les instructions définies dans le bloc situé après le mot-clé `else` sont exécutées.

De cette façon, un seul des deux blocs est exécuté, selon que la condition est vérifiée ou non.

De plus, cette condition fait intervenir des :

- opérateurs relationnels :

Opérateur	Signification pour des valeurs numériques	Signification pour des valeurs de type caractère
= =	égal	identique
<	inférieur strictement	plus petit dans l'ordre alphabétique
<=	inférieur ou égal	plus petit ou identique dans l'ordre alphabétique
>	supérieur strictement	plus grand dans l'ordre alphabétique
>=	supérieur ou égal	plus grand ou identique dans l'ordre
! =	différent	alphabétique différent

- opérateurs logiques :

Opérateur	Signification
!	NON logique
&&	ET logique
||	OU logique

Copyright © 2016 Eyrolles.

Lorsque plusieurs instructions `if-else` sont imbriquées les unes dans les autres, un `else` se rapporte toujours au dernier bloc `if` rencontré auquel un `else` n'a pas encore été attribué.

L'instruction `switch` (traduction : selon ou suivant) permet de programmer des choix multiples. Elle a pour syntaxe :

```
switch(valeur)      // le type de la variable est char ou int
{
   case étiquette :  // suite d'instructions
      break ;  // facultatif, pour sortir du bloc switch
   case étiquette :  // suite d'instructions
      break ;  // facultatif, pour sortir du bloc switch
   default       :  // suite d'instructions
}
```

La variable `valeur` est évaluée. Suivant cette évaluation, le programme recherche l'étiquette correspondant à la valeur évaluée et définie à partir des instructions `case étiquette` .

- Si le programme trouve une étiquette correspondant au contenu de la variable `valeur`, il exécute la ou les instructions qui suivent l'étiquette, jusqu'à rencontrer le mot-clé `break`.
- S'il n'existe pas d'étiquette correspondant à `valeur`, alors le programme exécute les instructions de l'étiquette `default`.

L'instruction `if-else` est utilisée lorsque l'une des conditions envisagées a une grande probabilité d'être satisfaite. Si toutes les conditions ont une même probabilité d'être réalisées, on utilise plutôt la structure `switch`.

Exercices

Comprendre les niveaux d'imbrication

Exercice 3.1 Exécutez à la main (c'est-à-dire ligne par ligne) ce programme. Pour cela, vous supposerez que la valeur saisie au clavier soit `4`. Quel est le résultat affiché ?

```
import java.util.*;
public class Racine
{
  public static void main (String [] parametre)
  {
    double  x, r ;
    Scanner lectureClavier = new Scanner(System.in);
    System.out.print("Entrer un chiffre :") ;
```

Copyright © 2016 Eyrolles.

```
    x = lectureClavier.nextDouble();
    if (x > = 0)
    {
      r = Math.sqrt(x) ;
    }
    else
    {
      r = Math.sqrt(-x) ;
    }
    System.out.print("Pour "+x+" Le resultat est: "+r) ;
  } // Fin du main ()
} // Fin de la Class Racine
```

Même question en supposant la valeur saisie égale à – 9.

Construire une arborescence de choix

Exercice 3.2 Reprenez et modifiez le programme `Maximum` donné dans ce chapitre, de façon qu'il affiche un message lorsque les deux valeurs saisies au clavier sont égales.

Exercice 3.3 Représentez graphiquement les choix arborescents suivants :

```
if (Condition 1)
  {
    if (Condition 2)
    {
      if (Condition 3)
      {
         instruction A
      }
    }
    else
    {
      instruction B
    }
  }
  else
  {
    instruction C
  }
```

Copyright © 2016 Eyrolles.

Exercice 3.4 Écrivez un programme qui résout les équations du second degré à l'aide de structures `if-else` imbriquées.

Soit l'équation $ax^2 + bx + c = 0$, où `a`, `b`, et `c` représentent les trois coefficients entiers de l'équation. Pour trouver les solutions réelles `x`, si elles existent :

a. Établissez l'arbre des choix associés :

```
1.   a = 0
  1.1.   b = 0
    1.1.1.  c = 0      tout réel est solution
    1.1.2.  c != 0     pas de solution
  1.2.  b != 0         une seule solution : x = - c / b ;
2.   a != 0
  2.1.  b² - 4ac >= 0   deux solutions :
      x1 = - b + Math.sqrt(b * b - 4 * a * c) / 2 * a ;
      x2 = - b - Math.sqrt(b * b - 4 * a * c) / 2 * a ;
  2.2.  b² - 4ac < 0    pas de solution dans les réels
```

b. Déterminez les différentes variables à déclarer.

c. À partir de l'arbre des choix, écrivez les instructions `if-else` suivies du test correspondant.

d. Placez dans chaque bloc `if` ou `else` les instructions de calcul et d'affichage appropriées.

e. Placez l'ensemble de ces instructions dans une fonction `main()` et une classe portant le nom `SecondDegre`.

Manipuler les choix multiples, gérer les caractères

Exercice 3.5 En utilisant la structure `switch`, écrire un programme qui simule une machine à calculer dont les opérations sont l'addition (+), la soustraction (–), la multiplication (*) et la division (/).

a. En cours d'exécution, le programme demande à l'utilisateur d'entrer deux valeurs numériques puis le caractère correspondant à l'opération à effectuer. Suivant le caractère entré (+ – * /) le programme affiche l'opération effectuée, ainsi que le résultat.

L'exécution du programme peut, par exemple, avoir l'allure suivante (les valeurs grisées sont celles saisies par l'utilisateur) :

```
Entrez la premiere valeur : 2
Entrez la seconde valeur : 3
Type de l'operation  (+, -, *, /) : *
Cette operation a pour resultat : 2 * 3 = 6
```

b. Après avoir écrit et exécuté le programme avec différentes valeurs, saisissez dans cet ordre les valeurs suivantes : 2, 0 puis /. Que se passe-t-il ? Pourquoi ?

c. Modifiez le programme de façon à ne plus rencontrer cette situation en cours d'exécution.

Copyright © 2016 Eyrolles.

Le projet : Gestion d'un compte bancaire

Accéder à un menu suivant l'option choisie

L'objectif est d'améliorer le programme réalisé à la fin du chapitre 2, « Communiquer une information », afin d'afficher chaque menu en fonction de l'option choisie par l'utilisateur.

a. Après l'affichage du menu principal, le programme teste la valeur entrée par l'utilisateur et affiche l'option correspondante. Sachant que toutes les options du menu principal ont une probabilité voisine ou équivalente d'être réalisées, quelle est la structure de test la plus appropriée ?

b. Modifiez le programme en fonction de la structure de test choisi, et placez les instructions d'affichage et de saisie dans les options correspondantes.

c. Pour l'option `1`, testez le type du compte afin de saisir le taux d'épargne.

d. Pour l'option `2`, demandez au programme de vérifier que le numéro du compte saisi par l'utilisateur existe, de façon à :

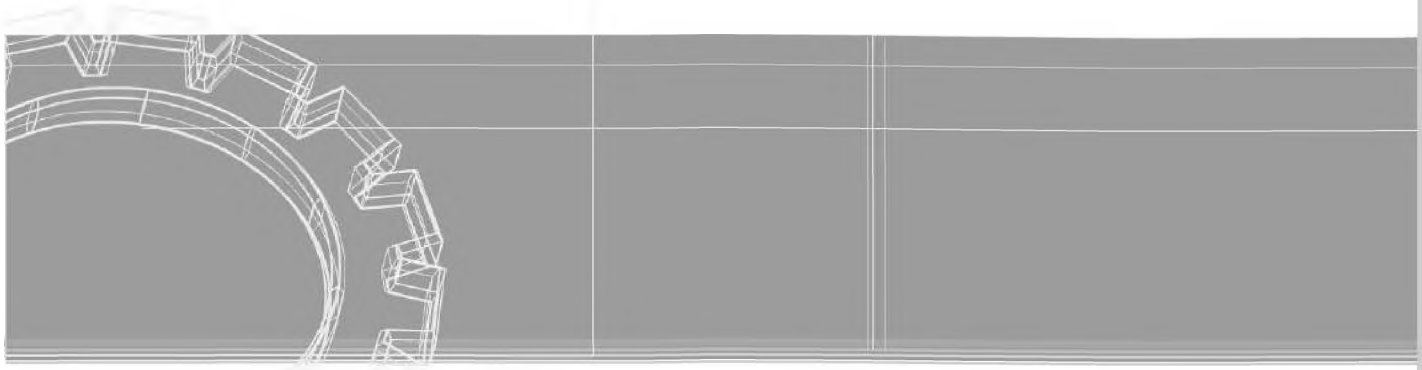
   - Afficher le numéro du compte, le type, la valeur initiale et son taux dans le cas d'un compte d'épargne, si le compte existe.
   - Afficher un message indiquant que le numéro du compte n'est pas valide, si le compte n'existe pas.

Copyright © 2016 Eyrolles.

Copyright © 2016 Eyrolles.

Chapitre 4

Faire des répétitions



La notion de répétition est une des notions fondamentales de la programmation. En effet, beaucoup de traitements informatiques sont répétitifs. Par exemple, la création d'un agenda électronique nécessite de saisir un nom, un prénom et un numéro de téléphone autant de fois qu'il y a de personnes dans l'agenda.

Dans de tels cas, la solution n'est pas d'écrire un programme qui comporte autant d'instructions de saisie qu'il y a de personnes mais de faire répéter par le programme le jeu d'instructions nécessaires à la saisie d'une seule personne. Pour ce faire, le programmeur utilise des instructions spécifiques, appelées structures de répétition, ou **boucles**, qui permettent de déterminer la ou les instructions à répéter.

Dans ce chapitre, nous abordons la notion de répétition à partir d'un exemple imagé (voir section « Combien de sucres dans votre café »).

Nous étudions ensuite les différentes structures de boucles proposées par le langage Java (sections « La boucle do...while », « La boucle while » et « La boucle for »). Pour chacune de ces structures, nous présentons et analysons un exemple afin d'examiner les différentes techniques de programmation associées aux structures répétitives.

Copyright © 2016 Eyrolles.

Combien de sucres dans votre café ?

Pour bien comprendre la notion de répétition ou de boucle, nous allons améliorer l'algorithme du café chaud sucré, de sorte que le programme demande à l'utilisateur de prendre un morceau de sucre autant de fois qu'il le souhaite. Pour cela, nous reprenons uniquement le bloc d'instructions *II Préparer le sucre* (voir, au chapitre 3, « Faire des choix », la section « L'algorithme du café chaud, sucré ou non »).

Instructions	Bloc d'instructions
`Si (café sucré)`	
1. Prendre une petite cuillère. 2. Poser la petite cuillère dans la tasse. 3. Prendre un morceau de sucre. 4. Poser le sucre dans la tasse.	*II Préparer le sucre*

L'exécution du bloc d'instructions *II Préparer le sucre* nous permet de mettre un seul morceau de sucre dans la tasse. Si nous désirons mettre plus de sucre, nous devons exécuter les instructions 3 et 4 autant de fois que nous souhaitons de morceaux de sucre. La marche à suivre devient dès lors :

```
Prendre une petite cuillère.
Poser la petite cuillère dans la tasse.
Début répéter :
    1. Prendre un morceau de sucre.
    2. Poser le sucre dans la tasse.
    3. Poser la question : "Souhaitez-vous un autre morceau de sucre ?"
    4. Attendre la réponse.
Tant que la réponse est OUI, retourner à Début répéter.
```

Analysons les résultats possibles de cette nouvelle marche à suivre :

- Dans tous les cas, nous prenons et posons une petite cuillère.
- Ensuite, nous entrons sans condition dans une structure de répétition.
- Nous prenons et posons un morceau de sucre, quelle que soit la suite des opérations. De cette façon, si nous sortons de la boucle, le café est quand même sucré.
- Puis le programme nous demande si nous souhaitons à nouveau un morceau de sucre.
- Si notre réponse est `OUI`, le programme retourne au début de la structure répétitive, place le sucre dans la tasse et demande de nouveau si nous souhaitons du sucre, etc.
- Si la réponse est négative, la répétition s'arrête, ainsi que la marche à suivre.

Copyright © 2016 Eyrolles.

Remarque

Pour écrire une boucle, il est nécessaire de déterminer où se trouve le début de la boucle et où se situe la fin (`Début répéter` et `Tant que` pour notre exemple).

La sortie de la structure répétitive est soumise à la réalisation ou non d'une condition (`la réponse` fournie est-elle affirmative ou non ?).

Le résultat du test de sortie de boucle est modifiable par une instruction placée à l'intérieur de la boucle (la valeur de la réponse est modifiée par l'instruction `4. Attendre la réponse`).

Question

Que se passe-t-il si l'on place les instructions :

```
Prendre une petite cuillère.
Poser la petite cuillère dans la tasse.
```

à l'intérieur de la structure de répétition :

```
Début répéter :
…
Tant que la réponse est OUI, retourner à Début répéter.
```

Réponse

Les deux instructions sont répétées autant de fois que l'on souhaite ajouter un morceau de sucre. Nous aurons donc autant de petites cuillères que de morceaux de sucre.

Dans le langage informatique, la construction d'une répétition ou boucle suit le même modèle. Dans le langage Java, il existe trois types de boucles, qui sont décrites par les constructions suivantes :

`do...while`	Faire... tant que
`while`	Tant que
`for`	Pour

Dans la suite de ce chapitre, nous allons, pour chacune de ces boucles :

- Étudier la syntaxe.
- Analyser les principes de fonctionnement.
- Donner un exemple qui introduise un concept fondamental de la programmation, à savoir le compteur de boucle, l'accumulation de valeurs ou la recherche d'une donnée parmi un ensemble d'informations.

La boucle do...while

La boucle `do...while` est une structure répétitive, dont les instructions sont exécutées avant même de tester la condition d'exécution de la boucle. Pour construire une telle structure, il est nécessaire de suivre les règles de syntaxe décrites ci-après.

Copyright © 2016 Eyrolles.

Syntaxe

La boucle do...`while` se traduit par les termes `faire... tant que`. Cette structure s'écrit de deux façons différentes en fonction du nombre d'instructions qu'elle comprend.

Dans le cas où une seule instruction doit être répétée, la boucle s'écrit de la façon suivante :

```
do
        une seule instruction ;
while (expression conditionnelle) ;
```

Si la boucle est composée d'au moins deux instructions, celles-ci sont encadrées par des accolades, ouvrante et fermante, de façon à déterminer où commence et se termine la boucle.

```
do {
        plusieurs instructions ;
} while (expression conditionnelle) ;
```

Principes de fonctionnement

Ainsi décrite, la boucle do...`while` s'exécute selon les principes suivants :

- Les instructions situées à l'intérieur de la boucle sont exécutées tant que l'expression conditionnelle placée entre parenthèses `()` est vraie.
- Les instructions sont exécutées au moins une fois, puisque l'expression conditionnelle est examinée en fin de boucle, après exécution des instructions.
- Si la condition mentionnée entre parenthèses reste toujours vraie, les instructions de la boucle sont répétées à l'infini. On dit que le programme « boucle ».
- Une instruction modifiant le résultat du test de sortie de boucle est placée à l'intérieur de la boucle, de façon à stopper les répétitions au moment souhaité.
- Observons qu'un point-virgule est placé à la fin de l'instruction `while (expression);`.

Un distributeur automatique de café

L'objectif de cet exemple est double : apprendre à construire une boucle do...`while` et étudier comment compter et accumuler des valeurs. Le comptage des valeurs, quelles qu'elles soient, est une technique très utilisée en informatique. Il existe deux façons de compter :

- Le **comptage** d'un certain nombre de valeurs. Par exemple, le programme compte le nombre de notes d'un étudiant.
- L'**accumulation** de valeurs. Le programme calcule la somme des notes d'un étudiant (les notes sont accumulées).

Le calcul de la moyenne des notes d'un étudiant s'effectue en divisant l'accumulation des notes par le nombre (comptage) de notes obtenues.

Pour bien comprendre ces différentes techniques, nous allons écrire un programme dont l'objectif est de simuler de façon simplifiée un distributeur automatique de café.

Copyright © 2016 Eyrolles.

Cahier des charges

Pour obtenir un café, l'utilisateur introduit un certain nombre de pièces de monnaie dans le distributeur. Pour simplifier, nous supposons que l'appareil n'accepte que les pièces de 5, 10 et 20 centimes d'euros (cents). Lorsqu'une pièce est introduite, le distributeur affiche la valeur totale engagée, ainsi que le nombre de pièces par catégorie (nombre de pièces de 5 cents, 10 cents et 20 cents). La machine prépare un café dès que la somme totale introduite vaut ou dépasse le prix du café. Nous prenons pour hypothèse que le prix d'un café est de 45 cents. La machine rend la monnaie, s'il y a lieu.

Après lecture et analyse du cahier des charges, nous observons que la démarche se déroule en trois temps.

1. Introduction une à une des pièces dans le distributeur.
2. À chaque pièce fournie, calcul et affichage :
   a. Du nombre de pièces de 10 cents, 20 cents et 5 cents.
   b. De la somme engagée.
3. Y a-t-il suffisamment d'argent ?
   a. Non, alors retourner en 1.
   b. Oui, alors préparer le café et rendre la monnaie.

Pour écrire le programme, nous allons nous attacher à résoudre, dans l'ordre, chacun de ces points.

1. Construire la boucle et introduire les pièces.

   Les points 1 et 3.a décrivent la structure de la boucle. L'introduction des pièces dans le distributeur est une opération répétitive, qui s'arrête lorsque l'utilisateur a placé suffisamment d'argent dans le distributeur, c'est-à-dire lorsque le montant total engagé vaut ou dépasse la somme de 45 cents. Par conséquent, l'allure générale de la structure répétitive est la suivante :

```
Début répéter
  Entrer une pièce de monnaie
  Compter la somme engagée
Tant que la somme engagée ne dépasse pas 45 cents, retourner à
Début répéter.
```

   En langage Java, cette structure est traduite en reprenant la syntaxe de la boucle `do...while`, c'est-à-dire par :

```
do // Début de boucle
{
   // Entrer les pièces de monnaie
   // Compter la somme engagée
}
while (somme engagée < 45 cents); // Fin de boucle
```

Copyright © 2016 Eyrolles.

De cette façon, la boucle est exécutée tant que la somme engagée est inférieure à 45 cents. Dès que cette somme vaut ou dépasse 45 cents, la condition `somme engagée < 45 cents` n'est plus vérifiée, et le programme sort de la boucle.

Ensuite, pour simuler l'introduction des pièces de monnaie dans le distributeur, le programme demande à l'utilisateur de saisir au clavier la valeur de chaque pièce entrée.

Pour en savoir plus Pour plus d'informations sur la saisie de valeurs au clavier, voir le chapitre 2, « Communiquer une information ».

Nous écrivons donc :

```
System.out.print("valeur de la piece entree : ");
pièce = lectureClavier.nextInt();
```

2. Compter le nombre de pièces et la somme totale engagée.

   Pour compter le nombre de pièces de 5 cents, 10 cents et 20 cents, le programme doit pouvoir distinguer les différentes pièces introduites. Pour cela, nous déclarons autant de variables qu'il y a de catégories de pièces, soit :

```
int nbPièce10C = 0, nbPièce20C= 0, nbPièce5C=0, pièce ;
int totalReçu = 0;
```

   Les variables dont le nom commence par `nb` représentent le nombre de pièces pour chacune des catégories. La variable `pièce` désigne, quant à elle, la valeur de la pièce saisie au clavier. Enfin, la variable `totalReçu` représente la somme totale engagée en cours d'exécution de la boucle. Ces variables sont déclarées de type `int`.

   a. Pour compter séparément les pièces de 5 cents, de 10 cents et de 20 cents, la meilleure méthode consiste à placer dans la boucle `do...while` une structure `switch` distinguant quatre cas :

```
switch (pièce)
   {
     case 5 :
       // Compter les pièces de 5 cents
       break;
     case 10 :
       // Compter les pièces de 10 cents
       break;
     case 20 :
       // Compter les pièces de 20 cents
       break;
     default :
       System.out.println ("Piece impossible");
 }
```

Copyright © 2016 Eyrolles.

Suivant la valeur de la pièce engagée, le programme compte le nombre de pièces, pour chacune des catégories en utilisant une instruction du type :

```
a = a + 1;
```

où `a` représente l'objet à compter. Si la variable `a` est initialisée à `0`, la nouvelle valeur de `a`, après affectation, vaut `1`.

Pour en savoir plus Pour plus d'informations sur ce mécanisme de calcul voir, au chapitre 1, « Stocker une information », la section « Quelques confusions à éviter ».

Placé dans une structure répétitive, le nombre d'objets représentés par `a` augmente de 1 à chaque tour de boucle. En informatique, on dit que `a` est **incrémenté** de 1. Pour compter le nombre de pièces de 5 cents, 10 cents et 20 cents, il suffit de remplacer la variable `a` par `nbPièce5C`, `nbPièce10C` ou `nbPièce20C`. Nous obtenons ainsi, pour chaque catégorie de pièces, les instructions suivantes :

```
nbPièce5C = nbPièce5C + 1;
nbPièce10C = nbPièce10C + 1;
nbPièce20C = nbPièce20C + 1;
```

b. Ces instructions sont ensuite placées dans les étiquettes `5`, `10` et `20` de la structure `switch`.

Pour en savoir plus Pour mieux comprendre l'évolution de la valeur de ces variables, reportez-vous à la section « Question Réponse » ci dessous.

Pour calculer la somme engagée à chaque pièce introduite, la technique est légèrement différente de la précédente. En effet, la somme engagée doit être augmentée non plus du nombre de pièces introduites mais de la **valeur** de la pièce introduite. L'incrément n'est plus de 1 mais de la valeur de la pièce. Comme la variable `pièce` représente la valeur de la pièce, l'instruction d'**accumulation** est la suivante :

```
totalReçu = totalReçu + pièce;
```

Ainsi, la variable `totalReçu`, initialisée à zéro, augmente progressivement de la valeur de chaque pièce engagée, par accumulation de la valeur précédente de `totalReçu` avec la valeur de la pièce entrée.

Ce calcul est réalisé quelle que soit la valeur de la pièce. Par conséquent, cette instruction est placée en dehors de la structure `switch`, mais, à l'intérieur de la boucle. Le montant total engagé est modifié chaque fois qu'une nouvelle pièce de 5 cents, 10 cents ou 20 cents est introduite.

Copyright © 2016 Eyrolles.

Pour éviter d'accumuler dans `totalReçu` la valeur d'une pièce non autorisée, nous devons modifier la valeur de la pièce dans l'étiquette `default` de la structure `switch` par l'instruction :

```
default :
        pièce = 0;
        System.out.println ("Piece impossible");
```

Lorsqu'une mauvaise pièce est introduite, la variable `pièce` prend la valeur `0`. De cette façon, l'instruction d'accumulation est réalisée, quelle que soit la valeur de la pièce, puisque la variable `totalReçu` n'est pas modifiée par l'accumulation d'une pièce valant `0` euro.

Pour en savoir plus Pour mieux comprendre l'évolution de la valeur de la variable `totalReçu` reportez-vous à la section « Question - Réponse» ci-après.

Une fois le nombre de pièces compté et le montant total calculé, le programme affiche les différentes valeurs à l'aide des instructions suivantes :

```
System.out.println("Vous avez entre  : ");
System.out.println("    " + nbPièce5C + " piece(s) de 5 cents");
System.out.println("    " + nbPièce10C + " piece(s) de 10 cents");
System.out.println("    " + nbPièce20C + " piece(s) de 20 cents");
System.out.println("Soit au total :  " + totalReçu + " cents");
```

L'ensemble de ces instructions est placé avant le test de sortie de boucle, puisque les valeurs calculées sont affichées chaque fois que l'utilisateur entre une pièce.

3. Y a-t-il suffisamment d'argent ?

   a. Non, alors retourner en 1.

   Il s'agit de déterminer la condition de sortie ou non de la boucle. Cette opération est décrite au point 1.

   Notons, cependant, que grâce à l'instruction d'accumulation

   ```
   totalReçu = totalReçu + pièce;
   ```

   la valeur de la variable `totalReçu` est augmentée à chaque tour de boucle. Par conséquent, le résultat de la condition de sortie de boucle (`totalReçu < 45`) ne reste pas toujours vrai. Le programme peut sortir de la boucle.

   b. Oui, alors préparer le café et rendre la monnaie.

   Lorsque l'utilisateur a entré suffisamment de pièces de monnaie, le programme affiche un message qui annonce que le café est prêt, à l'aide de l'instruction :

   ```
   System.out.println("Je vous verse 1 cafe ");
   ```

Copyright © 2016 Eyrolles.

Pour détecter un trop-perçu, le programme teste si `totalReçu` dépasse la valeur du prix du café. Si tel est le cas, il calcule la monnaie à rendre et affiche un message en conséquence. Ces actions sont réalisées par les instructions :

```
if (totalReçu > 45)
System.out.println("et vous rends : " + (totalReçu-45)
                                        + "cents ");
```

Exemple : code source

Pour obtenir un programme à part entière, l'ensemble des instructions développées au cours de la section précédente est à placer dans une fonction `main()` et une classe, comme ci-dessous :

```
import java.util.*;
public class CompteurMonnaie
{
 public static void main(String [] arg)
 {
  int nbPièce5C = 0, nbPièce10C = 0, nbPièce20C=0, pièce;
  int totalReçu = 0;
  Scanner lectureClavier = new Scanner(System.in);
  System.out.println("Pour obtenir un cafe, entrez au moins 45
                      cents");
  System.out.println("Je rends la monnaie ");
  do
  {
   System.out.print("valeur de la piece entree : ");
   pièce = lectureClavier.nextInt();
   switch (pièce)
   {
     case 5 :
       nbPièce5C = nbPièce5C + 1;
       break;
     case 10 :
       nbPièce10C = nbPièce10C + 1;
       break;
     case 20 :
       nbPièce20C = nbPièce20C + 1;
       break;
     default :
       pièce = 0;
       System.out.println ("Piece impossible");
   }
   totalReçu = totalReçu + pièce;
   System.out.println("Vous avez entre : ");
   System.out.println("     " + nbPièce5C + " piece(s) de 5 cents");
   System.out.println("     " + nbPièce10C + " piece(s) de 10 cents");
   System.out.println("     " + nbPièce20C + " piece(s) de 20 cents");
   System.out.println("Soit au total :  " + totalReçu + " cents");
```

Copyright © 2016 Eyrolles.

```
    } while (totalReçu < 45);
    System.out.println("Je vous verse 1 cafe ");
    if (totalReçu > 45)
     System.out.println("et vous rends : " + (totalReçu-45) + " cents ");
    }
}
```

Question Comment s'exécute le programme `CompteurMonnaie`, si l'utilisateur entre successivement les valeurs *10*, *30*, *20* et *20* ?

Réponse Le tableau d'évolution des variables créées en mémoire lors de l'exécution du programme est le suivant :

	pièce	nb Pièce5C	nb Pièce10C	nb Pièce20C	total Reçu
Valeurs initiales au début de l'exécution	-	0	0	0	0
Valeur de la piece entree : 10 Vous avez entre : 0 piece(s) de 5 cents 1 piece(s) de 10 cents 0 piece(s) de 20 cents Soit au total : 10 cents	10	0	1 car 0 + 1	0	2 car 0 + 10
Valeur de la piece entree : 30 Piece impossible Vous avez entre : 0 piece(s) de 5 cents 1 piece(s) de 10 cents 0 piece(s) de 20 cents Soit au total : 10 cents	30 0	0	1	0	10 car 10 + 0
Valeur de la piece entree : 20 Vous avez entre : 0 piece(s) de 5 cents 1 piece(s) de 10 cents 1 piece(s) de 20 cents Soit au total : 30 cents	20	0	1	1 car 0 + 1 = 1	30 car 10 + 20
Valeur de la piece entree : 20 Vous avez entre : 0 piece(s) de 5 cents 1 piece(s) de 10 cents 2 piece(s) de 20 cents Soit au total : 50 cents	20	0	1	2 car 1 + 1 = 2	50 car 30 + 20
Je vous verse 1 cafe et vous rends : 5 cents	20	0	1	2	50

Copyright © 2016 Eyrolles.

Question Que se passe-t-il si l'on supprime l'instruction :

```
pièce = 0;
```

dans le cas `default` ?

Réponse La valeur de la pièce saisie reste stockée dans la variable *pièce* (par exemple 30 cents pour l'exemple précédent) et l'instruction :

```
totalReçu = totalReçu + pièce;
```

a pour conséquence d'incrémenter la variable `totalReçu` de `30` alors que la pièce de 30 cents n'existe pas. En réinitialisant la variable `pièce` à 0, la variable `totalReçu` est incrémentée de `0`. Cela n'a donc plus d'incidence sur le résultat final.

La boucle while

Le langage Java propose une autre structure répétitive, analogue à la boucle `do...while`, mais dont la décision de poursuivre la répétition s'effectue en début de boucle. Il s'agit de la boucle `while`.

Syntaxe

La boucle `while` s'écrit de deux façons différentes, en fonction du nombre d'instructions qu'elle comprend.

Dans le cas où une seule instruction doit être répétée, la boucle s'écrit :

```
while (expression conditionnelle)
        une seule instruction ;
```

Si la boucle est composée d'au moins deux instructions, celles-ci sont encadrées par des accolades, ouvrante et fermante, de façon à déterminer où débute et se termine la boucle.

```
while (expression conditionnelle)
{
        plusieurs instructions ;
}
```

Principes de fonctionnement

Le terme `while` se traduit par `tant que`. La structure répétitive s'exécute selon les principes suivants :

- Tant que l'expression à l'intérieur des parenthèses reste vraie, la ou les instructions composant la boucle sont exécutées.

Copyright © 2016 Eyrolles.

- Le programme sort de la boucle dès que l'expression à l'intérieur des parenthèses devient fausse.
- Une instruction est placée à l'intérieur de la boucle pour modifier le résultat du test à l'entrée de la boucle, de façon à stopper les répétitions.
- Si l'expression à l'intérieur des parenthèses est fausse dès le départ, les instructions ne sont jamais exécutées.
- Observons qu'à l'inverse de la boucle `do...while`, il n'y a pas de point-virgule à la fin de l'instruction `while (expression)`.

Saisir un nombre entier au clavier

L'objectif de cet exemple est d'apprendre à écrire une boucle `while` et de comprendre comment réaliser la saisie d'un entier au clavier sans utiliser la classe `Scanner`.

Nous avons déjà noté (voir, au chapitre 2, « Communiquer une information », la section « La saisie de données ») que la fonction `System.in.read()` ne permettait de saisir qu'un seul caractère à la fois au clavier. Pour saisir un nombre composé de plusieurs chiffres ou un mot constitué de plusieurs caractères, nous devons faire appel à la fonction `System.in.read()` autant de fois qu'il y a de caractères à saisir.

Cette saisie de caractères est donc une opération répétitive, qui doit s'arrêter lorsque la valeur numérique ou le mot est entièrement entré. L'ordinateur n'est pas à même de déterminer quand la saisie est terminée. L'utilisateur confirme qu'il a fini d'entrer des valeurs en appuyant sur une touche caractéristique du clavier. Cette touche, utilisée pour passer à la ligne dans les logiciels de traitement de texte, est communément appelée la touche `Entrée`.

Notre but étant de saisir une valeur numérique entière, nous devons traduire l'ensemble des caractères saisis, de façon à les stocker non plus dans un `String` mais dans une variable de type `int`. Si cette traduction n'est pas réalisée, il n'est pas possible d'additionner ou de diviser les caractères lus à la manière des valeurs numériques. Par exemple, le fait d'additionner la suite de caractères 123 avec la valeur 4 a pour résultat 1234. Par contre, après traduction des caractères en valeur numérique, la même opération donne pour résultat 127.

Cahier des charges

Nous venons de l'observer, pour confirmer que nous n'avons plus de caractère à saisir, nous devons appuyer sur la touche `Entrée` du clavier. Pour saisir une valeur numérique entière, la liste des opérations s'exprime sous la forme de la structure répétitive suivante :

1. Tant que le caractère saisi n'est pas le caractère `Entrée` :
   a. Lire un caractère.

Copyright © 2016 Eyrolles.

b. Stocker le caractère lu dans un mot.

Retourner en 1.

2. Tous les caractères étant saisis, les traduire en un nombre entier.

Pour écrire le programme en langage Java, reprenons cette marche à suivre point par point.

1. La boucle `tant que` est traduite en Java par la construction suivante :

```
while (C != '\n')
{
 // Lire un caractère au clavier
 // Stocker le caractère dans un mot
}
```

En Java, le caractère `Entrée` est symbolisé par le caractère `'\n'` sur des ordinateurs de type Unix ou Macintosh. Sur un PC, la touche `Entrée` correspond à la série de caractères `'\r'` et `'\n'`. Afin de rendre compatible le programme avec tous les ordinateurs, nous allons tester la condition de sortie de boucle sur le caractère `'\n'`, puisque celui-ci est commun à tous les mondes, qu'ils soient Unix, Macintosh ou PC. De cette façon, en écrivant `while (C != '\n')`, où `C` représente le caractère lu, nous exprimons en langage informatique la phrase : tant que le caractère saisi n'est pas le caractère `Entrée`.

La première fois que le programme entre dans la boucle, aucun caractère n'a encore été saisi. Il est donc nécessaire d'initialiser la variable `C` à un caractère différent de `'\n'`, de façon à assurer que la condition d'entrée dans la boucle soit au moins vérifiée la première fois. Pour cela, nous déclarons `C` en début de programme, de la façon suivante :

```
char C ='\0';
```

Par cette instruction, nous initialisons la variable `C` au caractère nul (`'\0'`). Nous aurions pu l'initialiser à tout autre caractère à condition que celui-ci fût différent de `'\n'`. Le choix du caractère nul n'est ici réalisé que parce que, en général, les variables de type entier ou réel sont initialisées à `0` ou `0.0`. En Java, le caractère `'\0'` est l'équivalent de la valeur numérique nulle.

a. Pour lire un caractère au clavier, l'instruction est la suivante :

```
C = (char) System.in.read() ;
```

La fonction `System.in.read()` attend que l'utilisateur appuie sur une touche du clavier. Cela fait, elle retourne en résultat la valeur entière correspondant au caractère associé à la touche du clavier. Pour traduire cette valeur entière en code caractère, il est nécessaire de placer le cast `(char)` devant la fonction. De cette façon, la variable `C` contient le code Unicode du caractère saisi.

Copyright © 2016 Eyrolles.

b. Stocker le caractère lu dans un mot.

L'objectif est de lire plusieurs caractères d'affilée. Nous devons donc stocker dans une variable de type `String` chaque caractère au fur et à mesure de la saisie. Grâce au type `String`, plusieurs caractères peuvent être stockés sous un même nom de variable. La méthode consiste à accumuler dans une variable les valeurs lues, en utilisant l'instruction :

```
tmp = tmp + C;
```

Pour en savoir plus Le type `String` est détaillé, au chapitre 7, « Les classes et les objets », dans la section « La classe String, une approche vers la notion d'objet ».

Cette instruction permet d'accumuler les valeurs saisies en les plaçant les unes derrière les autres dans la variable `tmp`. En effet, lorsque deux caractères sont additionnés, ceux-ci sont placés dans la variable l'un après l'autre dans l'ordre d'exécution de l'opération. L'addition du caractère `'e'` et du caractère `'t'` a pour résultat le mot `et`. Dans le jargon informatique, l'addition de caractères est aussi appelée la **concaténation** de caractères.

En début de programme, la variable `tmp` ne doit pas contenir de caractère. Cela vient du fait que, la première fois qu'un caractère lu est placé dans la variable `tmp`, il doit correspondre au tout premier caractère du mot stocké dans la variable `tmp`. C'est pourquoi, la variable `tmp` doit être déclarée de la façon suivante (`""` correspondant à un mot vide de caractère) :

```
String tmp = "";
```

Lorsque, au final, l'utilisateur appuie sur la touche `Entrée` pour valider la fin de la saisie, le programme (sur PC) reçoit la suite de caractères `'\r'` et `'\n'`. La variable `tmp` contient en définitive la suite des caractères saisis, plus les caractères `'\r'` et `'\n'`. Or, nous souhaitons transformer cette suite de caractères en valeur numérique. Pour cela, nous devons éliminer les caractères `'\r'` et `'\n'`, qui empêchent cette transformation. L'accumulation des caractères ne se réalise donc qu'à la condition que le caractère saisi ne soit égal ni à `'\r'`, ni à `'\n'`.

Pour en savoir plus Pour plus d'informations, reportez-vous à la section «Traduire les caractères en un nombre entier », un peu plus loin dans ce chapitre.

Pour résumer, la boucle s'écrit :

```
String tmp = "";
char C ='\0';
while (C != '\n')
```

Copyright © 2016 Eyrolles.

```
{
 C = (char) System.in.read() ;
 if (C != '\r' && C != '\n') tmp = tmp + C;
}
```

Pour mieux comprendre en pratique le déroulement de cette boucle, examinons l'évolution des variables à partir d'un exemple. Nous supposons que l'utilisateur entre les caractères 2, 8 et Entrée.

	C	tmp	Explication
`String tmp = "";`	_	""	Initialisation.
`char C ='\0';`	\0	""	Initialisation.
`while (C != '\n')` `{`	\0	""	C étant initialisé au caractère '\0', C est différent du caractère '\n'. La condition placée entre () est vérifiée. Le programme entre dans la boucle.
`C = (char)System.in.read();`	2	""	Le programme attend la saisie d'une valeur au clavier. Nous supposons que le caractère saisi soit 2.
`if (C != '\r' && C != '\n')` `tmp = tmp + C;`	2	2	Le caractère C étant différent de '\r', la concaténation est exécutée. La variable tmp étant initialisée à la chaîne vide (""), l'opération "" + '2' stocke le caracère 2 en première position dans la variable tmp.
`}`	2	2	Fin de boucle. Le programme retourne en début de boucle.
`while (C != '\n')` `{`	2	2	La variable C contient la valeur 2. C est donc différent du caractère '\n'. La condition placée entre () est vérifiée. Le programme entre dans la boucle.
`C = (char)System.in.read();`	8	2	Nous entrons le caractère 8.
`if (C != '\r' && C != '\n')` `tmp = tmp + C;`	8	28	Le caractère C étant différent de '\r', l'opération '2' + '8' est exécutée et stocke le mot 28 dans la variable tmp.
`}`	8	28	Fin de boucle. Le programme retourne en début de boucle.

Copyright © 2016 Eyrolles.

	C	tmp	Explication
`while (C != '\n')` `{`	8	28	La variable C contient le caractère 8. C est donc différent du caractère `'\n'`. La condition placée entre `()` est vérifiée. Le programme entre dans la boucle.
`C=(char)System.in.read();`	\r	28	Nous appuyons sur la touche `Entrée`. Sur PC, le premier caractère entré est `'\r'`.
`if (C != '\r' && C != '\n')` `tmp = tmp+C;`	\r	28	C vaut `'\r'`. La condition n'étant pas vérifiée, il n'y a pas accumulation du caractère dans `tmp`.
`}`	\r	28	Fin de boucle. Le programme retourne en début de boucle.
`while (C != '\n')` `{`	\r	28	La variable C contient le caractère `\r`. C est donc différent du caractère `'\n'`. La condition placée entre `()` est vérifiée. Le programme entre dans la boucle.
`C = (char)System.in.read();`	\n	28	Le caractère suivant envoyé par la touche `Entrée` est `'\n'`.
`if (C != '\r' && C != '\n')` `tmp = tmp + C;`	\n	28	C vaut `'\n'`. La condition n'est pas vérifiée, et il n'y a pas accumulation du caractère dans `tmp`.
`}`	\n	28	Fin de boucle. Le programme retourne en début de boucle.
`while (C !='\n')` `{`	\n	28	La variable C contient le caractère `\n`. La condition placée entre `()` n'est plus vérifiée. Le programme sort de la boucle et passe à l'étape suivante.

2. Traduire les caractères en un nombre entier.

   Pour traduire un ensemble de caractères en une valeur numérique, le langage Java propose un certain nombre de fonctions. Dans notre cas, il s'agit de traduire un mot en une valeur entière de type `int`. La fonction Java `Integer.parseInt()` permet une telle traduction. L'instruction est la suivante :

   ```
   valeur = Integer.parseInt(tmp);
   ```

   `valeur` est une variable déclarée de type `int`, et `tmp` est le mot qui contient les caractères à traduire. La variable `tmp` ne doit contenir que des caractères représentant des chiffres.

Copyright © 2016 Eyrolles.

Si tel n'est pas le cas, le programme s'arrête avec un message d'erreur à l'exécution. Par exemple, si l'utilisateur entre le mot deux, au lieu du caractère 2, l'interpréteur Java affiche le message suivant :

```
java.lang.NumberFormatException : deux
at java.lang.Integer.parseInt (compiled Code)
```

Ce message indique que le format du nombre saisi ne correspond pas au format attendu par la fonction `Integer.parseInt()`. Nous aurions obtenu le même type d'erreur en stockant les caractères `'\r'` ou `'\n'` dans la variable `tmp`.

Pour finir, le programme affiche les différents résultats à l'aide de la fonction `System.out.println`. Cet affichage est réalisé à la fin du code source complet ci-dessous.

Exemple : code source complet

Pour obtenir un programme à part entière, l'ensemble des instructions développées au cours de la section précédente est à placer dans une fonction `main()` et une classe, comme ci-dessous :

```
public class LireUnEntier
{
  public static void main (String [] param) throws java.io.IOExcep-
tion
   {
    String tmp = "";
    char C= '\0';
    int valeur ;
    System.out.print("Entrez des chiffres et appuyez sur ");
    System.out.println("la touche Entree, pour valider la saisie :
");
    while (C != '\n')
    {
     C = (char) System.in.read() ;
     if (C != '\r' && C != '\n') tmp = tmp + C;
    }
    System.out.println("Vous avez entre : " + tmp);
    valeur = Integer.parseInt(tmp);
    System.out.println("C'est a dire : " + valeur + " en entier");
   } // Fin du main ()
} // Fin de la Class LireUnEntier
```

Notez l'expression `throws IOException` placée juste après l'en-tête de la fonction `main()`. La présence de cette expression est obligatoire pour toutes les méthodes qui manipulent des opérations d'entrée-sortie. Succinctement, cette clause indique au compilateur que la

Copyright © 2016 Eyrolles.

méthode ainsi définie est susceptible de traiter ou de propager une éventuelle erreur, du type `IOException`, qui pourrait apparaître en cours d'exécution.

Pour en savoir plus Pour plus de précisions sur la notion d'exception, voir la section « Gérer les exceptions », en fin du chapitre 10, « Collectionner un nombre indéterminé d'objets ».

Question Que se passe-t-il si l'utilisateur entre au clavier la valeur suivante (valeur grisée) :

```
Entrez des chiffres et appuyez sur la touche Entree, pour valider
la saisie : 28
```

Réponse Le programme affiche les messages suivants :

```
Vous avez entre : 28
C'est a dire : 28 en entier
```

La première valeur `28` affichée est un mot. L'addition de cette valeur avec le nombre `4` a pour résultat `284`. La deuxième valeur affichée est un nombre, et la même addition a pour résultat `32`.

Question Que se passe-t-il si l'utilisateur entre au clavier la valeur suivante (valeur grisée) :

```
Entrez des chiffres et appuyez sur la touche Entree, pour valider
la saisie : trois
```

Réponse Le programme affiche les messages suivants :

```
java.lang.NumberFormatException : trois
at java.lang.Integer.parseInt (compiled Code)
```

Le mot `trois` n'est pas un nombre mais un mot sans signification particulière pour l'ordinateur ; l'interpréteur Java ne peut traduire ce mot en un nombre entier.

Question Que se passe-t-il si l'utilisateur entre au clavier la valeur suivante (valeur grisée) :

```
Entrez des chiffres et appuyez sur la touche Entree, pour valider
la saisie : 2.5
```

Réponse Le programme affiche les messages suivants :

```
java.lang.NumberFormatException: 2,5
at java.lang.Integer.parseInt (compiled Code)
```

Le mot `2.5` n'a pas le format d'un nombre entier mais d'un nombre réel. La fonction `Integer.parseInt()` ne peut le traduire en un nombre entier.

Copyright © 2016 Eyrolles.

La boucle for

L'instruction `for` permet d'écrire des boucles dont on connaît à l'avance le nombre d'itérations (de tours) à exécuter. Elle est équivalente à l'instruction `while` mais est plus simple à écrire.

Syntaxe

La boucle `for` s'écrit elle aussi de deux façons différentes en fonction du nombre d'instructions qu'elle comprend.

Dans le cas où une seule instruction doit être répétée, la boucle s'écrit :

```
for (initialisation; condition; incrément)
  une seule instruction ;
```

Si la boucle est composée d'au moins deux instructions, celles-ci sont encadrées par deux accolades, ouvrante et fermante, de sorte à déterminer où débute et se termine la boucle.

```
for (initialisation; condition; incrément)
{
  plusieurs instructions ;
}
```

Les termes `initialisation`, `condition` et `incrément` sont des instructions séparées obligatoirement par des points-virgules (`;`). Ces instructions définissent une variable, ou indice, qui contrôle le bon déroulement de la boucle. Ainsi :

- `initialisation` permet d'initialiser la variable représentant l'indice de la boucle (exemple : `i = 0`, `i` étant l'indice). Elle est la première instruction exécutée, à l'entrée de la boucle.
- `condition` définit la condition à vérifier pour continuer à exécuter la boucle (exemple : `i < 10`). Elle est examinée avant chaque tour de boucle, y compris au premier tour de boucle.
- `incrément` est l'instruction qui permet de modifier le résultat du test précédent en augmentant ou diminuant la valeur de la variable testée. L'incrément peut être augmenté ou diminué de *N*. *N* est appelé le **pas d'incrémentation** (exemple : `i = i + 2`). Cette instruction est exécutée à la fin de chaque tour de boucle.

Remarque

Avec la version 1.5 du compilateur (`jdk1.5.0`), le langage Java propose une nouvelle syntaxe pour la boucle `for`, qui simplifie le parcours de liste de valeurs (tableaux, collection...). Cette syntaxe est étudiée plus précisément au cours du chapitre 9, « Collectionner un nombre fixe d'objets », section « Manipuler un tableau ».

Copyright © 2016 Eyrolles.

Principes de fonctionnement

Les boucles `for` réalisent un nombre précis de boucles dépendant de la valeur initiale, de la valeur finale et du pas d'incrémentation. Voyons sur différents exemples comment ces boucles sont exécutées :

`int i;` `char c;`	Valeur initiale	Valeur finale	Pas d'incrémentation	Nombre de boucles	Valeurs prises par `i` ou `c`
`for (i = 0; i < 5; i = i +1)`	0	4	1	5	0, 1, 2, 3, 4
`for (i = 4; i <= 12; i = i + 2)`	4	12	2	5	4, 6, 8, 10, 12
`for (c = 'a'; c < 'f'; c = c + 1)`	'a'	'e'	1	5	a, b, c, d, e
`for (i = 5; i > 0 ; i = i - 1)`	5	1	−1	5	5, 4, 3, 2, 1

Remarques

Le nombre de tours est identique dans chacune de ces boucles, malgré une définition différente pour chacune des instructions de contrôle.

L'écriture de l'instruction `incrément`, qui augmente ou diminue de 1 la variable de contrôle de la boucle, peut être simplifiée. En effet, par convention, l'instruction `i = i + 1` s'écrit plus simplement `i++`, et l'instruction `i--` a le même résultat que l'instruction `i = i - 1`.

Rechercher le code Unicode d'un caractère de la table ASCII

L'objectif de cet exemple est d'apprendre à construire une boucle `for` et de s'initier à la recherche d'information dans un ensemble de données. Pour cela, nous allons écrire un programme qui recherche dans la table Unicode le code d'un caractère ASCII donné par l'utilisateur. Cette recherche s'effectue en comparant chaque caractère de la table Unicode au caractère ASCII saisi.

Cahier des charges

La méthode est la suivante :

1. Lire le caractère ASCII dont on souhaite connaître le code Unicode.
2. Pour chaque caractère de la table Unicode :

   Si le caractère Unicode est identique au caractère choisi, afficher son code Unicode.

Reprenons chaque point, pour le traduire en un programme Java.

1. Pour lire au clavier le caractère dont on souhaite connaître le code Unicode, les instructions sont les suivantes :

```
System.out.println("Quel caractere recherchez-vous : ");
recherche = lectureClavier.next().charAt(0);
```

   Où la variable `recherche` est déclarée de type `char`.

Copyright © 2016 Eyrolles.

Pour en savoir plus Pour plus d'informations sur la saisie de valeurs au clavier, voir le chapitre 2, « Communiquer une information ».

2. Le programme parcourt la table Unicode caractère par caractère et recherche le caractère souhaité. Cette opération est répétitive et s'exécute sur les 128 premières valeurs de la table Unicode, puisque nous ne recherchons que les codes Unicode des caractères de la table ASCII.

Pour en savoir plus Pour plus d'informations sur la table Unicode, voir, au chapitre 1, « Stocker une information », la section « Catégorie caractère ».

Pour parcourir cette table, la solution est d'utiliser une boucle `for`, dont la valeur de l'indice varie de 1 en 1, dans l'intervalle `[0, 127]`. Cette boucle s'écrit :

```
for (i = 0; i <= 127; i++)
```

La variable `i`, déclarée de type `int`, représente l'indice du caractère dans la table Unicode. Il y a équivalence entre l'indice et le caractère. En effet, un caractère ASCII est défini à partir d'une valeur numérique.

La seule différence entre une valeur numérique et un caractère provient du type de codage utilisé pour les représenter l'un et l'autre. Pour connaître le caractère correspondant à cet indice, la méthode consiste à transformer la valeur de l'indice en un code caractère par l'intermédiaire du cast `(char)`. Ainsi, l'instruction :

```
atrouver = (char) i;
```

transforme l'indice `i` de la table Unicode en son code caractère. La variable `atrouver`, déclarée de type `char`, prend la valeur de ce code.

Connaissant le caractère à rechercher ainsi que le code caractère de chaque caractère de la table Unicode, il suffit de les comparer pour savoir s'ils sont identiques ou non. L'instruction s'écrit sous la forme du test suivant :

```
if (atrouver == recherche)
```

Si le caractère Unicode est identique au caractère choisi, le programme affiche son code Unicode à l'aide des instructions :

```
System.out.print("le code Unicode de " + atrouver);
System.out.println(" est \\u00" + Integer.toString(i,16));
```

Rappelons que pour la table ASCII, le code Unicode d'un caractère s'obtient en plaçant derrière les caractères `\u00`, la valeur hexadécimale de la position du caractère dans la table Unicode. Pour afficher ce code, nous devons donc traduire la variable `i` (qui correspond

Copyright © 2016 Eyrolles.

à la position du caractère dans la table Unicode) en valeur hexadécimale. Cette traduction est réalisée par la fonction :

```
Integer.toString(valeur entière, base)
```

qui transforme le paramètre `valeur entière` en une chaîne de caractères suivant le codage donné par le paramètre `base`. Si `valeur entière` représente l'indice `i` et que `base` prend la valeur 16, nous obtenons la valeur hexadécimale de la position du caractère trouvé.

La suite des caractères `\u00` placée dans la fonction `System.out.println` est considérée comme une séquence particulière puisqu'elle permet l'affichage des caractères spéciaux. Pour annuler le caractère spécifique de cette séquence, il est nécessaire de placer un premier `\` devant `\u00`.

Exemple : code source complet

Pour obtenir un programme à part entière, l'ensemble des instructions développées au cours de la section précédente est à placer dans une fonction `main()` et une classe, comme ci-dessous :

```
import java.util.*;
public class QuelUnicode
{
 public static void main (String [] parametre)
 {
  int i;
  char recherche, atrouver;
  Scanner lectureClavier = new Scanner(System.in);
  System.out.println("Quel caractere recherchez-vous : ");
  recherche = lectureClavier.next().charAt(0);
  for (i = 0; i <= 127; i++)
  {
   atrouver = (char) i;
   if (atrouver == recherche)
   {
    System.out.print("le code Unicode de " + atrouver);
    System.out.println(" est \\u00" + Integer.toString(i,16));
   } // Fin du if
  } // Fin du for
 } // Fin du main()
}// Fin de QuelUnicode
```

Copyright © 2016 Eyrolles.

Résultat de l'exécution

Les valeurs grisées correspondent aux valeurs saisies par l'utilisateur.

```
Quel caractere recherchez-vous : &
le code Unicode de & est \u0026
Quel caractere recherchez-vous : {
le code Unicode de { est \u007b
```

Remarque

Il n'est pas possible de retrouver le code Unicode d'un caractère accentué, c'est-à-dire d'un caractère dont la valeur se situe au-delà de l'indice `127` de la table Unicode. En effet, comme nous avons pu le remarquer à la section « Les caractères spéciaux » du chapitre 2, « Communiquer une information », il existe plusieurs formes d'encodage des caractères spéciaux. La transformation d'une valeur numérique en `char` par l'intermédiaire d'un simple `cast` n'est pas suffisante pour déterminer le code Unicode des caractères spéciaux.

Quelle boucle choisir ?

Chacune des trois boucles étudiées dans ce chapitre permet de répéter un ensemble d'instructions. Cependant, les différentes propriétés de chacune d'entre elles font que le programmeur utilisera un type de boucle plutôt qu'un autre, suivant le problème à résoudre.

Choisir entre une boucle do...while et une boucle while

Les boucles `do...while` et `while` se ressemblent beaucoup dans leur syntaxe, et il paraît parfois difficile au programmeur débutant de choisir l'une plutôt que l'autre.

Notons cependant que la différence essentielle entre ces deux boucles réside dans la position du test de sortie de boucle. Pour la boucle `do...while`, la sortie de boucle s'effectue en fin de boucle, alors que, pour la boucle `while`, la sortie de boucle se situe dès l'entrée de la boucle.

De ce fait, la boucle `do...while` est plus souple à manipuler, les instructions qui la composent étant exécutées au moins une fois, quoi qu'il arrive. Pour la boucle `while`, il est nécessaire de veiller à l'initialisation de la variable figurant dans le test d'entrée de boucle, de façon à être sûr d'exécuter au moins une fois les instructions composant la boucle.

Certains algorithmes demandent à ne jamais répéter, sous certaines conditions, un ensemble d'instructions. Dans de tels cas, la structure `while` est préférable à la structure `do...while`.

Copyright © 2016 Eyrolles.

Choisir entre la boucle for et while

Les boucles `for` et `while` sont équivalentes. En effet, en examinant les deux boucles du tableau ci-dessous.

La boucle `for`	La boucle `while`
`int i;`	`int i = 0`
`for (i = 0; i <= 10; i = i+1 )`	`while (i <= 10)`
`{`	`{`
`}`	`  i = i+1;`
	`}`

Nous constatons que, pour chacune d'entre elles, la boucle débute avec `i = 0`, puis, tant que `i` est inférieur ou égal à `10`, `i` est incrémenté de `1`.

Malgré cette équivalence, pour choisir entre une boucle `for` et une boucle `while`, observons que :

- La boucle `for` est utilisée quand on connaît à l'avance le nombre d'itérations à exécuter.
- La boucle `while` est employée lorsque le nombre d'itérations est laissé au choix de l'utilisateur du programme ou déterminé à partir du résultat d'un calcul réalisé au cours de la répétition.

Résumé

En langage Java, il existe trois types de structures pour réaliser des répétitions. Elles sont décrites par les instructions : `do...while`, `while` et `for`.

- La boucle `do...while` (faire... tant que) permet d'exécuter les instructions situées dans le bloc défini par des `{}`, tant que l'expression conditionnelle placée entre `()` est vraie.

```
do {
  plusieurs instructions ;
} while (expression) ;
```

Les instructions sont exécutées au moins une fois puisque l'expression conditionnelle est examinée en fin de boucle, après exécution des instructions.

Copyright © 2016 Eyrolles.

- La boucle `while` (tant que) permet d'exécuter les instructions situées dans le bloc défini par `{}`, tant que l'expression conditionnelle placée entre `()` est vraie.

```
while (expression)
{
  plusieurs instructions ;
}
```

L'expression conditionnelle étant examinée en début de boucle, les instructions situées dans le bloc peuvent ne pas être exécutées si la condition n'est pas vérifiée dès le début.

- La boucle `for` permet d'écrire des boucles dont on connaît à l'avance le nombre d'itérations à exécuter. Elle est équivalente à l'instruction `while` mais est plus simple à écrire.

```
for (initialisation ; condition ; incrément)
{
  plusieurs instructions ;
}
```

Les termes `initialisation`, `condition` et `incrément` sont des instructions séparées obligatoirement par des points-virgules (`;`). Ces instructions définissent un indice qui contrôle le bon déroulement de la boucle. Ainsi :

- `initialisation` permet d'initialiser la variable représentant l'indice de la boucle.
- `condition` définit la condition à vérifier pour continuer à exécuter la boucle.
- `incrément` permet d'augmenter ou de diminuer de *N* la valeur de la variable représentant l'indice de la boucle. *N* est appelé le pas d'incrémentation.

À partir des structures répétitives nous avons également abordé la notion de comptage de valeurs, c'est-à-dire :

- Le **comptage** d'un certain nombre de valeurs (par exemple, compter le nombre de notes d'un étudiant). Pour cela, il suffit d'employer une variable entière initialisée à `0` avant d'entamer la boucle. La variable augmente de `1` à l'intérieur de la boucle à l'aide de l'instruction `i = i + 1` (en supposant que `i` soit notre variable compteur). On dit alors que la variable `i` est incrémentée de `1`.
- L'**accumulation** de valeurs (par exemple, faire la somme des notes d'un étudiant). Cette technique est réalisée à l'aide d'une variable entière initialisée à `0` avant d'entamer la boucle. La variable augmente de la valeur de la variable à accumuler (de la valeur de la note, par exemple), à l'intérieur de la boucle. Cette augmentation s'effectue à l'aide de l'instruction `s = s + valeur`, en supposant que `s` soit notre variable d'accumulation et `valeur` la variable représentant la valeur à accumuler.

Signalons, pour finir, que l'instruction `i++` est l'équivalent simplifié de `i = i + 1`, tandis que `i--` est l'équivalent simplifié de `i = i - 1`.

Copyright © 2016 Eyrolles.

Exercices

Comprendre la boucle do...while

Exercice 4.1 Afin d'exécuter le programme suivant :

```
import java.util.*;
public class Exercice1
{
 public static void main (String [] argument)
 {
  int a,b,r;
  Scanner lectureClavier = new Scanner(System.in);
  System.out.print("Entrer un entier : ");
  a = lectureClavier.nextInt();
  System.out.print("Entrer un entier : ");
  b = lectureClavier.nextInt();
  do
  {
    r = a%b;
    a = b;
    b = r;
   } while (r !=0 );
  System.out.println("Le resultat est " + a);
 }
}
```

a. Examinez le code source (programme), repérez les instructions concernées par la boucle répétitive, et déterminez les instructions de début et fin de boucle.

b. Quelle est l'instruction qui permet de modifier le résultat du test de sortie de boucle ?

c. En supposant que l'utilisateur entre les valeurs `30` et `42`, exécutez le programme à la main (pour vous aider, construisez le tableau d'évolution de chaque variable déclarée).

d. En supposant que l'utilisateur entre les valeurs `35` et `6`, exécutez le programme à la main (pour vous aider, construisez le tableau d'évolution de chaque variable déclarée).

e. Quel est le calcul réalisé par ce programme ?

Exercice 4.2 En utilisant une boucle `do...while`, écrire un programme qui demande la saisie d'une valeur, tant que celle-ci n'est pas comprise entre `0` et `100`.

Copyright © 2016 Eyrolles.

Apprendre à compter, accumuler et rechercher une valeur

Exercice 4.3 Écrivez en français, en faisant ressortir la structure répétitive de la marche à suivre, le programme résolvant les quatre points suivants :

a. Lire un nombre quelconque de valeurs entières non nulles. La saisie des valeurs se termine lorsqu'on entre la valeur 0.

b. Afficher la plus grande des valeurs.

c. Afficher la plus petite des valeurs.

d. Calculer et afficher la moyenne de toutes les valeurs.

Traduisez la marche à suivre précédente en un programme Java. Utilisez pour cela une boucle `do...while`.

Pour en savoir plus Pour trouver la plus grande ou la plus petite valeur, vous pouvez vous aider de l'exemple « Rechercher le plus grand de deux éléments », décrit chapitre 3, section « Faire des choix ».

Comprendre la boucle while, traduire une marche à suivre en programme Java

Exercice 4.4 Écrivez un programme `Devinette`, qui tire un nombre au hasard entre 0 et 10 et demande à l'utilisateur de trouver ce nombre. Pour ce faire, la méthode est la suivante :

a. Tirer au hasard un nombre entre 0 et 10.

b. Lire un nombre.

c. Tant que le nombre lu est différent du nombre tiré au hasard :

– Lire un nombre.

– Compter le nombre de boucle.

d. Afficher un message de réussite ainsi que le nombre de boucles.

Reprenez chaque point énoncé ci-dessus, et traduisez-le en langage Java. Notez que, pour tirer un nombre au hasard entre 0 et 10, l'instruction s'écrit :

```
i = (int) (10*Math.random());
```

où `i` est une variable entière qui reçoit la valeur tirée au hasard.

Exercice 4.5 Déclarez toutes les variables utilisées dans le programme précédent en veillant à ce qu'elles soient bien initialisées. Placez les instructions dans une fonction `main()` et une classe `Devinette`.

Copyright © 2016 Eyrolles.

Exercice 4.6 Lorsque le programme `Devinette` fonctionne bien, modifiez-le de façon à ce que :

a. Les valeurs tirées au hasard soient comprises entre 0 et 50.

b. Un message d'erreur s'affiche si la réponse est mauvaise.

c. Le programme indique si la valeur saisie au clavier est plus grande ou plus petite que la valeur tirée au hasard.

d. À titre de réflexion : comment faut-il s'y prendre pour trouver la valeur en donnant le moins de réponses possibles ?

Comprendre la boucle for

Exercice 4.7 Afin d'exécuter le programme suivant :

```
import java.util.*;
public class Exercice7
{
 public static void main (String [] parametre)
 {
  long i, b = 1;
  int a;
  Scanner lectureClavier = new Scanner(System.in);
  System.out.println("Entrer un entier :");
  a = lectureClavier.nextInt();
  for (i = 2; i <= a; i++)
    b = b * i;
  System.out.println("Le resultat vaut " + b);
 }
}
```

a. Examinez le programme, repérez les instructions concernées par la boucle répétitive, et déterminez les instructions de début et fin de boucle.

b. Quelle est la valeur initiale de `i` et quelle est sa valeur en sortie de boucle ? Combien de boucles sont réalisées ?

c. Quelle est l'instruction qui permet de modifier le résultat du test de sortie de boucle ?

d. En supposant que l'utilisateur entre la valeur `6`, exécutez le programme à la main (pour vous aider, construisez le tableau d'évolution de chaque variable déclarée).

e. Quel est le calcul réalisé par ce programme ?

Exercice 4.8 En utilisant une boucle `for`, écrivez un programme qui affiche l'alphabet, d'abord à l'endroit, puis à l'envers, après un passage à la ligne.

Copyright © 2016 Eyrolles.

Le projet : Gestion d'un compte bancaire

Rendre le menu interactif

Une fois l'affichage du menu réalisé à partir de l'énoncé donné à la fin du chapitre 3, « Faire des choix », le programme exécuté donne à choisir parmi les cinq options suivantes :

```
1. Création d'un compte
2. Affichage d'un compte
3. Créer une ligne comptable
4. Sortir
5. De l'aide
```

Votre choix :

Si l'utilisateur choisit l'option `1`, le programme lui demande de saisir les données nécessaires à la création du compte (type, numéro, valeur initiale, etc.). Une fois les données saisies, le programme s'arrête. Il n'est pas possible de choisir, par exemple, l'option `2` pour afficher les valeurs saisies à l'étape précédente.

Pour remédier à cette situation, il est nécessaire de placer les instructions concernées à l'intérieur d'une boucle, de façon à voir réapparaître le menu une fois l'option réalisée. Pour cela, vous devez :

a. Écrire en français la structure répétitive, afin de déterminer la condition de sortie de boucle.

b. Choisir la structure répétitive parmi les trois proposées par le langage Java.

c. Traduire la marche à suivre en programme Java, en prenant soin d'initialiser la variable de contrôle de la boucle et en insérant, à l'intérieur de la boucle, toutes les instructions nécessaires à l'affichage du menu.

Copyright © 2016 Eyrolles.

Copyright © 2016 Eyrolles.

Partie II

Initiation à la programmation orientée objet

Copyright © 2016 Eyrolles.

Copyright © 2016 Eyrolles.

Chapitre 5

De l'algorithme paramétré à l'écriture de fonctions

L'étude des chapitres précédents montre qu'un programme informatique est constitué d'instructions élémentaires (affectation, comparaison ou encore répétition) et de sous-programmes (calcul de la racine carrée, affichage de données), appelés fonctions ou encore méthodes.

Ces instructions sont de nature suffisamment générale pour s'adapter à n'importe quel problème. En les utilisant à bon escient, il est possible d'écrire des programmes informatiques simples mais d'une grande utilité.

Dans le cadre du développement de logiciels de grande envergure, les programmeurs souhaitent aussi définir leurs propres instructions, adaptées au problème qu'ils traitent. Pour cela, les langages de programmation offrent la possibilité de créer des fonctions spécifiques, différentes des fonctions prédéfinies par le langage.

Pour comprendre l'intérêt des fonctions, nous analysons d'abord le concept d'algorithme paramétré à partir d'un exemple imagé.

Ensuite, nous étudions la bibliothèque de fonctions mathématiques définie dans le langage Java (section « Des fonctions Java prédéfinies »). Cette étude montre les principes d'utilisation de ces fonctions et explique comment élaborer et construire vos fonctions (section « Construire ses propres fonctions »).

Pour finir, nous examinons comment la construction et l'utilisation de fonctions font évoluer la structure générale d'un programme (section « Les fonctions au sein d'un programme Java »).

Copyright © 2016 Eyrolles.

Algorithme paramétré

Certains algorithmes peuvent être appliqués à des problèmes voisins en modifiant simplement les données pour lesquels ils ont été construits. En faisant varier certaines valeurs, le programme fournit un résultat différent du précédent. Ces valeurs, caractéristiques du problème à traiter, sont appelées paramètres du programme.

Pour comprendre concrètement ce concept, nous allons reprendre l'algorithme du café chaud pour le transformer en un algorithme qui nous permettra de faire du thé ou du café chaud.

Faire un thé chaud, ou comment remplacer le café par du thé

Faire un café chaud ou faire un thé chaud est une opération à peu près semblable. En reprenant la liste de toutes les opérations nécessaires à la réalisation d'un café chaud, nous constatons qu'en remplaçant simplement le mot café par le mot thé, nous obtenons du thé chaud.

Instructions	Bloc d'instructions
0. Prendre une cafetière.	
1. Poser la cafetière sur la table.	
3. Prendre un filtre.	
4. Verser le **thé** dans le filtre.	
5. Prendre de l'eau.	
6. Verser l'eau dans la cafetière.	
7. Brancher la cafetière.	*Préparer le **thé***
8. Allumer la cafetière.	
9. Attendre que le **thé** soit prêt.	
10. Prendre une tasse.	
11. Poser la tasse sur la table.	
12. Éteindre la cafetière.	
13. Verser le **thé** dans la tasse.	

Cette recette n'est certes pas traditionnelle, mais elle a le mérite d'être pédagogiquement simple. Pour faire du café ou du thé, il suffit d'employer la même recette ou méthode, en prenant comme ingrédient du café ou du thé, selon notre choix.

Dans le monde réel, le fait de remplacer un ingrédient par un autre ne pose pas de difficultés particulières. Dans le monde informatique, c'est plus complexe. En effet, l'ordinateur ne fait qu'exécuter la marche à suivre fournie par le programmeur. Dans notre cas, pour avoir du café ou du thé, le programmeur doit écrire la marche à suivre pour chacune des boissons. La tâche est fastidieuse, puisque chacun des programmes se ressemble, tout en étant différent sur un détail (café ou thé).

Copyright © 2016 Eyrolles.

Définir les paramètres

Pour éviter d'avoir à recopier chaque fois des marches à suivre qui ne diffèrent que sur un détail, l'idée est de construire un algorithme général. Cet algorithme ne varie qu'en fonction d'ingrédients déterminés, qui font que le programme donne un résultat différent.

En généralisant l'algorithme du thé ou du café chaud, on exprime une marche à suivre permettant de réaliser une boisson chaude. Pour obtenir un résultat différent (café ou thé), il suffit de définir comme paramètre de l'algorithme l'ingrédient, café ou thé, à choisir.

La marche à suivre s'écrit en remplaçant les mots café ou thé par le mot **ingrédient**.

Instructions	Nom du bloc d'instructions
0. Prendre une cafetière. 1. Poser la cafetière sur la table. 2. Prendre **ingrédient**. 3. Prendre un filtre. 4. Verser ingrédient dans le filtre. 5. Prendre de l'eau. 6. Verser l'eau dans la cafetière. 7. Brancher la cafetière. 8. Allumer la cafetière. 9. Attendre que **ingrédient** soit prêt. 10. Prendre une tasse. 11. Poser la tasse sur la table. 12. Éteindre la cafetière. 13. Verser **ingrédient** dans la tasse.	*Préparer (**ingrédient**)*

Faire du café équivaut donc à exécuter le bloc d'instructions *Préparer* (*ingrédient*) en utilisant comme ingrédient du café. L'exécution du bloc *Préparer* (*le café*) a pour conséquence de réaliser les instructions 2, 4, 9 et 13 du bloc d'instructions avec comme ingrédient du café. L'instruction 2, par exemple, s'exécute en remplaçant le terme `ingrédient` par `le café`. Au lieu de lire `prendre ingrédient`, il faut lire `prendre le café`.

De la même façon, faire du thé revient à exécuter le bloc d'instructions *Préparer* (`le thé`). Le paramètre `ingrédient` correspond ici au thé, et les instructions 2, 4, 9 et 13 sont exécutées en conséquence.

Suivant la valeur prise par le paramètre `ingrédient`, l'exécution de cet algorithme fournit un résultat différent. Ce peut être du café ou du thé.

Donner un nom au bloc d'instructions

Nous constatons qu'en paramétrant un algorithme, nous n'avons plus besoin de recopier plusieurs fois les instructions qui le composent pour obtenir un résultat différent.

Copyright © 2016 Eyrolles.

En donnant un nom au bloc d'instructions correspondant à l'algorithme général *Préparer()*, nous définissons un sous-programme capable d'être exécuté autant de fois que nécessaire. Il suffit pour cela d'appeler le sous-programme par son nom.

De plus, grâce au paramètre placé entre les parenthèses qui suivent le nom du sous-programme, la fonction s'exécute avec des valeurs différentes, modifiant de ce fait le résultat.

Remarque

Un algorithme paramétré est défini par :
- un nom ;
- un ou plusieurs paramètres.

En fin d'exécution, il fournit un résultat, qui diffère suivant la valeur du ou des paramètres.

Dans le langage Java, les algorithmes paramétrés s'appellent des **fonctions** ou encore des **méthodes**. Grâce à elles, il est possible de traduire un algorithme paramétré en programme informatique. Avant d'examiner comment écrire ces algorithmes en langage Java, nous allons tout d'abord étudier les fonctions prédéfinies du langage Java, de façon à mieux comprendre comment elles s'utilisent.

Des fonctions Java prédéfinies

Un grand nombre de programmes informatiques font appel à des calculs mathématiques simples, tels que le calcul d'un sinus ou d'une racine carrée. Pour trouver la valeur d'un sinus, par exemple, le programmeur n'a pas, fort heureusement, à réécrire pour chaque programme l'algorithme mathématique du calcul d'un sinus. Les fonctions mathématiques sont déjà programmées.

Le langage Java propose un ensemble de fonctions prédéfinies, mathématiques ou autres, très utiles, comme nous le verrons au cours des chapitres suivants. Notre objectif n'est pas de décrire l'intégralité des fonctions disponibles, car ce seul manuel n'y suffirait pas. Nous souhaitons faire comprendre la manipulation de ces fonctions. Pour ce faire, nous allons étudier une partie de la bibliothèque mathématique de Java, appelée `Math`, et déterminer ensuite les principes généraux d'utilisation des fonctions.

La bibliothèque Math

La bibliothèque mathématique du langage Java est composée d'un ensemble de fonctions prédéfinies, qui permettent de calculer toutes sortes d'équations mathématiques. Parmi ces fonctions, se trouvent les fonctions trigonométriques (sinus, cosinus, tangente, etc.), logarithmiques, d'arrondis, de calcul de puissances ou de racines carrées.

Ces fonctions sont regroupées dans la bibliothèque de programmes `Math`. Le nom de chaque fonction débute toujours par le terme `Math`, suivi d'un point puis du nom de la fonction.

Copyright © 2016 Eyrolles.

Ce nom commence toujours par une minuscule. Voici une liste partielle des fonctions qui composent la bibliothèque `Math` :

Fonctions trigonométriques

Opération mathématique	Fonction Java
Calculer le cosinus d'un angle (radian)	`Math.cos()`
Calculer le sinus d'un angle (radian)	`Math.sin()`
Calculer la tangente d'un angle (radian)	`Math.tan()`

Fonctions logarithmiques

Opération mathématique	Fonction Java
Calculer le logarithme d'une valeur	`Math.log()`
Calculer l'exponentielle d'un nombre	`Math.exp()`

Calcul d'arrondis

Opération	Fonction Java
Arrondir à l'entier inférieur	`Math.floor()`
Arrondir à l'entier supérieur	`Math.ceil()`

Autres calculs mathématiques

Opération mathématique	Fonction Java
Calculer la racine carrée d'un nombre	`Math.sqrt()`
a^b (a puissance b)	`Math.pow()`
$\|a\|$ (valeur absolue de a)	`Math.abs()`

Divers

Opération	Fonction Java
Trouver la plus grande de deux valeurs	`Math.max()`
Trouver la plus petite de deux valeurs	`Math.min()`
Tirer un nombre au hasard entre 0 et 1	`Math.random()`

Copyright © 2016 Eyrolles.

Exemples d'utilisation

Ces fonctions s'utilisent en plaçant dans le programme Java le nom d'appel de la fonction. Voici en exemple un programme qui utilise l'ensemble des fonctions décrites ci-dessus :

Exemple : code source complet

```
import java.util.*;
public class FonctionMathématique
{
 public static void main(String [] argument)
 {
  double résultat, a, b;
  Scanner lectureClavier = new Scanner(System.in);
  System.out.print("Entrez une premiere valeur :");
  a = lectureClavier.nextDouble();
  System.out.print("Entrez une seconde valeur :");
  b = lectureClavier.nextDouble();
  résultat = Math.cos(a) ;
  System.out.println("Cos(" + a + ") = " + résultat);
  résultat = Math.sin(a) ;
  System.out.println("Sin(" + a + ") = " + résultat);
  résultat = Math.tan(a) ;
  System.out.println("Tan(" + a + ") = " + résultat);
  résultat = Math.log(a) ;
  System.out.println("Log(" + a + ") = " + résultat);
  résultat = Math.exp(a) ;
  System.out.println("Exp(" + a + ") = " + résultat);
  résultat = Math.floor(a) ;
  System.out.println("Floor(" + a + ") = " + résultat);
  résultat = Math.ceil(a) ;
  System.out.println("Ceil(" + a + ") = " + résultat);
  résultat = Math.sqrt(a) ;
  System.out.println("Sqrt(" + a + ") = " + résultat);
  résultat = Math.pow(a,b) ;
  System.out.println("Pow(" + a + ", " + b +") = " + résultat);
  résultat = Math.abs(a) ;
  System.out.println("Abs(" + a + ") = " + résultat);
  résultat = Math.max(a,b) ;
  System.out.println("Max(" + a + ", " + b + ") = " + résultat);
  résultat = Math.min(a,b) ;
  System.out.println("Min(" + a + ", " + b + ") = " + résultat);
  résultat = Math.random() ;
  System.out.println("Random() = " + résultat);
 }
}
```

Copyright © 2016 Eyrolles.

Une fois les instructions de ce programme compilées, l'interpréteur Java les exécute une à une.

Question Qu'affiche le programme `FonctionMathématique` si l'utilisateur entre les valeurs `0.1` et `2` ?

Réponse L'exécution du programme réalise l'affichage suivant :
Les caractères grisés sont des valeurs choisies par l'utilisateur.

```
Entrez une premiere valeur : 0.1
Entrez une seconde valeur : 2
Cos(0.1) = 0.9950041652780257
Sin(0.1) = 0.09983341664682815
Tan(0.1) = 0.10033467208545055
Log(0.1) = -2.3025850929940455
Exp(0.1) = 1.1051709180756477
Floor(0.1) = 0.0
Ceil(0.1) = 1.0
Sqrt(0.1) = 0.31622776601683794
Pow(0.1, 2.0) = 0.01
Abs(0.1) = 0.1
Max(0.1, 2.0) = 2.0
Min(0.1, 2.0)= 0.1
Random() = 0.6993848420032578
```

Principes de fonctionnement

L'étude de ce programme met en évidence plusieurs aspects importants concernant l'utilisation des fonctions et leur mode de fonctionnement.

Le nom des fonctions

- Le nom de chaque fonction est défini par le langage Java. Pour connaître le nom des différentes fonctions proposées par le langage Java, il est nécessaire de consulter l'aide en ligne du compilateur ou le site Internet de Sun (voir l'annexe « Guide d'installations », section « Installer la documentation en ligne »), ou encore des livres plus spécifiques sur le langage Java et les bases de données ou les réseaux.

Remarque L'exécution d'une fonction passe par l'écriture, dans une instruction, du nom de la fonction choisie, suivi de paramètres éventuels placés entre parenthèses.

Copyright © 2016 Eyrolles.

Mémoriser le résultat d'une fonction

Pour mémoriser le résultat du calcul, la fonction est placée dans une instruction d'affectation. La fonction, située à droite du signe =, est exécutée en premier. Après quoi, la variable située à gauche du signe = récupère la valeur calculée lors de l'exécution de la fonction.

Pour en savoir plus Pour plus d'informations, voir, au chapitre 1, « Stocker une information », la section « Rôle et mécanisme de l'affectation ».

Dans notre exemple, toutes les fonctions de la bibliothèque `Math` fournissent en résultat une valeur numérique de type `double`. En conséquence, la variable `résultat`, qui récupère le résultat de chaque fonction, est déclarée de type `double`.

Les paramètres d'une fonction

Les fonctions possèdent zéro, un, voire deux paramètres. Ainsi :

- La fonction `Math.random()` ne possède pas de paramètre. Cette fonction donne en résultat une valeur au hasard, comprise entre 0.0 et 1.0, indépendamment de toute condition. Aucun paramètre n'est donc nécessaire à sa bonne marche.
- Signalons que même si la fonction n'a pas de paramètre, il reste nécessaire de placer des parenthèses, ouvrante puis fermante, derrière le nom d'appel de la fonction. Si aucune parenthèse n'est placée, le compilateur ne considère pas le terme `Math.random` comme une fonction mais comme un nom de variable.

Remarque Toute fonction possède dans son nom d'appel des parenthèses, ouvrante puis fermante.

- La fonction `Math.sqrt()` ne comporte qu'un seul paramètre, puisqu'elle calcule la racine carrée d'un seul nombre à la fois. Il est possible de placer entre parenthèses une expression mathématique plutôt qu'un paramètre. Ainsi, l'expression `Math.sqrt(b*b - 4*a*c)` permet le calcul de la racine carrée du discriminant d'une équation du second degré.

  Observons que le paramètre placé entre parenthèses dans la fonction `Math.sqrt()` est de type `double`. De cette façon, il est possible de calculer la racine carrée de tout type de valeur numérique, les types `byte`, `short`, `int` ou `long` se transformant sans difficulté en type `double`.

Pour en savoir plus Pour plus d'informations, voir, au chapitre 1, « Stocker une information », la section « La transformation de types ».

Copyright © 2016 Eyrolles.

Il n'est pas permis de placer en paramètre un caractère, une suite de caractères ou un booléen. Par exemple, le fait d'écrire `Math.sqrt("Quatre")` entraîne une erreur en cours de compilation, l'ordinateur ne sachant pas transformer le mot « Quatre » en la valeur numérique 4 (message d'erreur : `Incompatible type for method. Can't convert java.lang.String to double`) .

Remarque Dans l'appel de la fonction, le type des paramètres doit être respecté, sous peine d'obtenir une erreur de compilation.

- La fonction `Math.pow(a,b)` possède deux paramètres pour calculer a^b (`a` à la puissance `b`). Ces paramètres sont séparés par une virgule. Si les valeurs `a` et `b` sont inversées dans l'appel de la fonction (`Math.pow(b,a)`), le calcul effectué a pour résultat b^a (`b` à la puissance `a`).

Remarque Dans l'appel de la fonction, l'ordre des paramètres doit être respecté, sous peine d'obtenir un résultat différent de celui attendu.

Les fonctions étudiées dans cette section sont des fonctions prédéfinies par le langage Java. Le programmeur les utilise en connaissant le résultat qu'il souhaite obtenir. Les programmes ainsi écrits sont constitués d'instructions simples et d'appels à des fonctions connues du langage Java.

Le langage Java offre aussi au programmeur la possibilité d'écrire ses propres fonctions de façon à obtenir différents programmes adaptés au problème qu'il doit résoudre. Nous étudions cette technique à la section qui suit.

Construire ses propres fonctions

Une fonction développée par un programmeur s'utilise de la même façon qu'une fonction prédéfinie. Elle s'exécute en plaçant l'instruction d'appel à la fonction dans le programme. Cette étape est décrite à la section « Appeler une fonction ».

Pour que l'ordinateur puisse lire et exécuter les instructions composant la fonction, il convient de définir cette dernière, c'est-à-dire d'écrire une à une les instructions qui la composent. Plusieurs étapes sont nécessaires à cette définition. Nous les étudions à la section « Définir une fonction ».

Pour mieux cerner les difficultés liées à ces opérations, nous allons prendre comme exemple la création d'une fonction qui calcule le périmètre d'un cercle de rayon quelconque.

Copyright © 2016 Eyrolles.

Appeler une fonction

Toute fonction possède un nom d'appel, qui permet de l'identifier. Ce nom est choisi de façon à représenter et résumer tout ce qui est réalisé par son intermédiaire. Dans notre exemple, nous devons calculer le périmètre d'un cercle. Nous appelons donc la fonction qui réalise ce calcul, c'est-à-dire `périmètre()`.

Remarque D'une manière générale, une fonction représente une action. C'est pourquoi le choix d'un verbe comme nom de fonction permet de mieux symboliser les opérations réalisées. Ici, le terme `périmètre()` n'est pas un verbe, mais il faut comprendre par `périmètre()` l'action de calculer le périmètre.

Le nom de la fonction `périmètre()` étant défini, nous souhaitons calculer le périmètre d'un cercle dont la valeur du rayon est saisie au clavier. Pour cela, observons le programme qui calcule la racine carrée d'un nombre saisi au clavier :

```
double résulat, a;
Scanner lectureClavier = new Scanner(System.in);
System.out.print("Entrez une valeur :");
a = lectureClavier.nextDouble();
résultat = Math.sqrt(a) ;
System.out.println("Sqrt(" + a + ") = " + résultat);
```

L'instruction `résultat = Math.sqrt(a) ;` calcule la racine carrée du nombre `a`, dont la valeur a été saisie au clavier à l'instruction précédente. Elle place ensuite le résultat de ce calcul dans la variable `résultat`.

En modifiant le nom d'appel de la fonction `Math.sqrt()` par `périmètre()`, nous obtenons un programme qui appelle la fonction `périmètre()` et qui, par conséquent, calcule le périmètre d'un cercle dont la valeur du rayon `a`, est saisie au clavier. La valeur du périmètre est placée dans la variable `résultat` par l'intermédiaire du signe d'affectation =.

Pour notre exemple, le programme d'appel à la fonction `périmètre()` s'écrit :

```
public static void main(String [] parametre)
{
 // Déclaration des variables
 double résultat ;
 int valeur ;
 Scanner lectureClavier = new Scanner(System.in);
 System.out.print("Valeur du rayon : ");
 valeur = lectureClavier.nextInt();
 résultat = périmètre (valeur);
 System.out.print("rayon = " + valeur + " perimetre = " + résultat);
}
```

Copyright © 2016 Eyrolles.

Question Que se passe-t-il si l'on veut compiler ce programme (après l'avoir inséré dans une classe) ?

Réponse Le compilateur affiche le message d'erreur suivant :

```
cannot resolve symbol,
symbol  : method périmètre  (int)
```

En effet, la fonction `périmètre()` n'est pas encore définie ; le compilateur n'est donc pas en mesure de comprendre à quoi correspond le terme `périmètre()`.

Le programme ainsi écrit permet de calculer le périmètre d'un cercle de rayon donné, à la seule condition de définir la fonction `périmètre()` dans le programme. En effet, cette fonction n'est pas prédéfinie dans le langage Java, et il est nécessaire de détailler les instructions qui la composent.

Définir une fonction

La définition d'une fonction fournit à l'ordinateur les instructions à exécuter lors de l'appel de la fonction. Cette opération passe par les étapes suivantes :

- déterminer les instructions composant la fonction ;
- associer le nom de la fonction aux instructions ;
- établir les paramètres utiles à l'exécution de la fonction ;
- préciser le type de résultat fourni par la fonction.

De façon à mieux comprendre le rôle de chacune de ces étapes, définissons la fonction qui calcule le périmètre d'un cercle de rayon quelconque.

Déterminer les instructions composant la fonction

Pour sélectionner les instructions utiles au calcul du périmètre d'un cercle, reprenons le programme `Cercle`.

Pour en savoir plus Voir, au chapitre introductif, « Naissance d'un programme », la section « Un premier programme en Java ».

```
import java.util.*;
public class Cercle
{
  public static void main(String [] argument)
  {
   // Déclaration des variables
   double r, p ;
```

Copyright © 2016 Eyrolles.

```
    Scanner lectureClavier = new Scanner(System.in);
    // Afficher le message "Valeur du rayon : " à l'écran
    System.out.print("Valeur du rayon : ") ;
    // Lire au clavier une valeur, placer cette valeur dans la variable r
    r = lectureClavier.nextDouble() ;
    // Calculer la circonférence en utilisant la formule consacrée
    p = 2*Math.PI*r ;
    // Afficher le résultat
    System.out.print("Le cercle de rayon "+ r +" a pour
                     perimetre : "+ p);
  }
}
```

Nous avons observé, lors de la mise en œuvre d'algorithmes paramétrés, que la marche à suivre décrivant l'algorithme devait être la plus générale possible (voir la section « Définir les paramètres »). C'est pourquoi, pour notre cas, seules les instructions :

```
// Déclaration des variables
   double r, p ;
// Calculer la circonférence en utilisant la formule consacrée
   p = 2*Math.PI*r ;
```

sont utilisées dans la fonction de calcul du périmètre d'un cercle. Les instructions relatives à la demande de saisie d'une valeur au clavier ne sont pas à placer dans la fonction. Pour vous en convaincre, observez que l'ordinateur, à l'appel de la fonction `Math.sqrt()`, ne demande pas de valeur à saisir. Il ne fait que calculer la racine carrée d'une valeur passée en paramètre.

Les instructions ainsi choisies sont placées dans ce que l'on appelle, dans le jargon informatique, le **corps de la fonction**, et ce de la façon suivante :

```
// Définition du corps de la fonction
{ // début de la fonction
  double p, r;
  p = 2 * Math.PI * r;
} // fin de la fonction
```

Le corps de la fonction est déterminé par les accolades `{` et `}`. Les instructions qui le composent sont ici des déclarations de variables et des instructions d'affectation. Dans d'autres cas, peuvent aussi figurer des instructions de test, de répétition, etc.

Associer le nom aux instructions

Une fois écrit le corps de la fonction, il est nécessaire de l'associer au nom d'appel de la fonction.

Le nom d'une fonction est lié au bloc d'instructions qui la compose, grâce à un **en-tête de fonction**. Ce dernier a pour forme

```
public static type nomdelafonction (paramètres)
```

Copyright © 2016 Eyrolles.

L'en-tête d'une fonction permet de préciser :

- Le nom de la fonction (pour notre exemple le `nomdelafonction` est `périmètre`).
- Les *`paramètres`* éventuels de la fonction.
- Le *`type`* de résultat fourni par la fonction.

Les mots-clés `public static` sont à placer pour l'instant obligatoirement devant le type de résultat de la fonction.

Pour en savoir plus Nous expliquons la présence de ces termes à la section « Collectionner un nombre fixe d'objets » du chapitre 9, « La ligne de commande », car ils sont liés aux concepts de la programmation objet.

L'en-tête d'une fonction se place, comme son nom l'indique, au-dessus du corps de la fonction. Pour notre exemple, il se place de la façon suivante :

```
// En-tête de la fonction
public static type périmètre (paramètres)
{ // début de la fonction
  double p, r;
  p = 2 * Math.PI * r;
} // fin de la fonction
```

De cette façon, le corps de la fonction est associé au nom `périmètre()`. À l'appel du nom de la fonction `périmètre()`, l'ordinateur exécute les instructions placées dans le corps de la fonction.

Établir les paramètres utiles

Comme nous venons de le voir, le périmètre du cercle est calculé à partir du rayon, dont la valeur est saisie avant l'appel de la fonction. La valeur du rayon est placée en paramètre de la fonction, comme lors du calcul de la racine carrée d'un nombre.

Le rayon du cercle est considéré comme le paramètre de la fonction `périmètre()`, et l'en-tête de la fonction s'écrit comme suit :

```
public static type périmètre (int r)
```

Comme la variable `r` est déclarée à l'intérieur des parenthèses de la fonction `périmètre()`, elle est considérée par le compilateur Java comme étant le paramètre de la fonction `périmètre()`. L'instruction de déclaration, située dans le corps de la fonction, doit être ainsi modifiée :

```
double p;
```

Copyright © 2016 Eyrolles.

La variable `r` est déclarée dans l'en-tête de la fonction, et elle ne peut donc être déclarée une deuxième fois à l'intérieur de la fonction, sous peine de provoquer une erreur de compilation (message d'erreur : `variable 'r' is already defined in this method`).

Remarque Le paramètre `r` est aussi appelé **paramètre formel**. Il prend la forme (la valeur) de la variable donnée au moment de l'appel de la fonction.

Pour bien comprendre cela, rappelons-nous de l'algorithme du café ou du thé chaud, dans lequel nous avons utilisé une variable `ingrédient` prenant la forme de café ou de thé suivant ce que l'on souhaitait obtenir. Ici, `r` prend la valeur de la variable `valeur` lors de l'appel de la fonction `résultat = périmètre(valeur)` depuis la fonction `main()`.

Remarque Le paramètre `valeur` fourni lors de l'appel de la fonction `périmètre()` est appelé **paramètre réel** ou encore **paramètre effectif**. C'est la valeur de ce paramètre qui est transmise au paramètre formel lors de l'appel de la fonction.

Question Que se passe-t-il au moment de la compilation, si l'on écrit l'en-tête de la fonction `périmètre()` de la façon suivante :

```
public static double périmètre ()
```

Réponse Le compilateur affiche le message d'erreur suivant :

```
cannot resolve symbol variable r
```

En effet, l'instruction :

```
p = 2 * Math.PI * r;
```

se situe à l'intérieur de la fonction `périmètre()`. Il est donc nécessaire de déclarer une variable `r`. Ce qui n'est pas fait, puisque `r` n'est déclaré ni à l'intérieur de la fonction, ni comme paramètre de la fonction.

Préciser le type de résultat fourni

Une fois le périmètre calculé grâce à l'instruction :

```
p = 2 * Math.PI * r;
```

la valeur contenue dans la variable `p` doit être transmise et placée dans la variable `résultat`, déclarée dans le programme décrit à la section « Appeler une fonction » de ce chapitre. Pour ce faire, les deux opérations suivantes sont à réaliser :

- Placer une instruction `return`, suivie de la variable contenant le résultat en fin de fonction. Pour notre cas :

```
return p;
```

Copyright © 2016 Eyrolles.

À la lecture de cette instruction, le programme sort de la fonction `périmètre()` et transmet la valeur contenue dans la variable `p` au programme qui a appelé la fonction `périmètre()`.

- Spécifier le type de la valeur retournée dans l'en-tête de la fonction. Pour notre exemple, la valeur retournée est contenue dans la variable `p` de type `double`. C'est pourquoi l'en-tête de la fonction s'écrit :

```
public static double périmètre (int r)
```

De cette façon, le compilateur sait, à la seule lecture de l'en-tête, que la fonction transmet un résultat de type `double`.

La fonction `périmètre()` s'écrit en résumé de la façon suivante :

```
public static double périmètre (int r)
{
  double p;
  p = 2 * Math.PI * r;
  return p;
}
```

Dans notre exemple, la fonction `périmètre()` utilise un seul paramètre et retourne un résultat numérique. Dans d'autres situations, le nombre de paramètres peut varier, et les fonctions peuvent avoir soit aucun, soit plusieurs paramètres. De la même façon, une fonction peut ne pas retourner de résultat.

Question Que se passe-t-il au moment de la compilation, si l'on écrit l'en-tête de la fonction `périmètre()` de la façon suivante :

```
public static int périmètre (int r)
```

Réponse Le compilateur affiche le message d'erreur suivant :

```
possible loss of precision
found   : double
required: int
```

En effet, le résultat retourné est stocké dans la variable `p` qui est de type `double`, alors que le type de retour précisé dans l'en-tête est `int`. Passer d'un type `double` à un type `int` entraîne une perte de précision qui peut gêner la bonne marche du programme.

Copyright © 2016 Eyrolles.

Les fonctions au sein d'un programme Java

Avec les fonctions, nous voyons apparaître la notion de fonctions **appelées** et de programmes **appelant** des fonctions.

Dans notre exemple, la fonction `périmètre()` est appelée par la fonction `main()`. Cette dernière est considérée par l'ordinateur comme étant le programme principal (le terme anglais *main* se traduit par principal). En effet, la fonction `main()` est la première fonction exécutée par l'ordinateur au lancement d'un programme Java.

Toute fonction peut appeler ou être appelée par une autre fonction. Ainsi, rien n'interdit que la fonction `périmètre()` soit appelée par une autre fonction que la fonction `main()`.

Seule la fonction `main()` ne peut pas être appelée par une autre fonction du programme. En effet, la fonction `main()` n'est exécutée qu'une seule fois, et uniquement par l'interpréteur Java, lors du lancement du programme.

Comment placer plusieurs fonctions dans un programme

Les fonctions sont des programmes distincts les uns des autres. Elles sont en outre définies séparément les unes des autres. Pour exécuter un programme constitué de plusieurs fonctions, il est nécessaire, pour l'instant, de les regrouper dans un même fichier, une même classe.

Pour en savoir plus Voir, au chapitre 7, « Les classes et les objets », la section « Compilation et exécution d'une application multifichier ».

Pour des raisons pédagogiques, les fonctions `main()` et `périmètre()` ont été présentées séparément. En réalité, ces deux fonctions sont placées à l'intérieur de la même classe `Cercle` (définie notamment au chapitre introductif, « Naissance d'un programme »).

Le programme prend la forme suivante :

```
import java.util.*;
public class Cercle  // Le fichier s'appelle Cercle.java
{
public static void main(String [] arg)
  {
   // Déclaration des variables
   int valeur ;
   double résultat ;
```

Copyright © 2016 Eyrolles.

```
    Scanner lectureClaviervaleur = lectureClavier.nextInt();
    résultat = périmètre (valeur) ;
    System.out.print("rayon = " + valeur + " perimetre = " + résultat);
   } // fin de main()

   public static double périmètre (int r)
   {
   double p ;
   p = 2 * Math.PI * r ;
   return p ;
   } // fin de périmètre()
} //fin de class Cercle
```

En examinant la structure générale de ce programme, nous observons qu'il existe deux blocs d'instructions séparés, nommés `main()` et `périmètre()`. Ces deux blocs sont placés à l'intérieur d'un bloc représentant la classe `Cercle`, comme illustré à la figure 5-1.

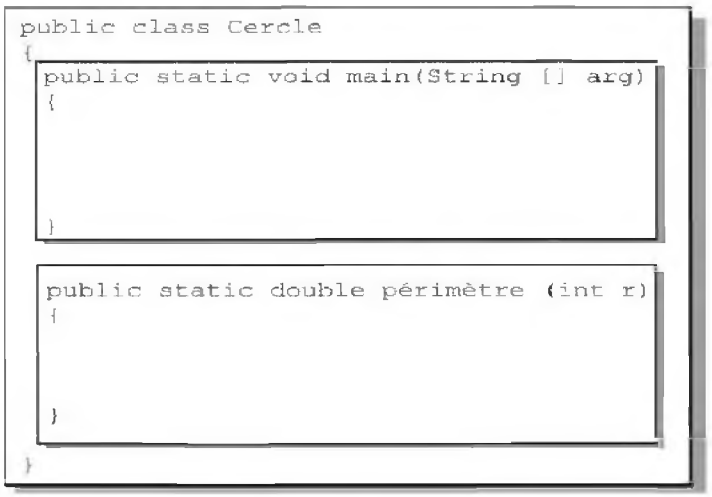


Figure 5-1 *Les fonctions main() et périmètre(), à l'intérieur de la classe Cercle.*

Nous observons que la structure de la fonction `périmètre()` est très voisine de celle de la fonction `main()`. Elle est constituée d'un en-tête, suivi d'un corps, formé d'un bloc défini par des accolades, ouvrante et fermante.

Notons, pour finir, que la fonction `main()` est ici placée avant la fonction `périmètre()` mais qu'il est aussi permis de l'écrire après. L'ordre d'apparition des fonctions dans une classe importe peu et est laissé au choix du programmeur.

Copyright © 2016 Eyrolles.

Les différentes formes d'une fonction

Nous l'avons déjà observé (voir la section « Principes de fonctionnement » de ce chapitre), les fonctions peuvent posséder zéro, un, voire plusieurs paramètres de différents types. De la même façon, elles peuvent fournir ou non un résultat. Suivant les cas, leur définition varie légèrement.

Fonction avec résultat

Comme nous l'avons observé lors de la définition de la fonction `périmètre()`, toute fonction fournissant un résultat possède un `return` placé dans le corps de la fonction. De plus, l'en-tête de la fonction possède obligatoirement un type, qui correspond au type du résultat retourné.

Si une fonction retourne en résultat une variable de type `int`, son en-tête s'écrit `public static int nomdelafonction()`.

Remarque

Une fonction ne retourne qu'une et une seule valeur. Il n'est donc pas possible d'écrire l'instruction `return` sous la forme `return a,b ;` pour retourner deux valeurs au programme appelant. Dans un tel cas, le compilateur détecte une erreur du type : « `';' expected` ».

Lorsqu'une fonction fournit plusieurs résultats, la transmission des valeurs ne peut se réaliser par l'intermédiaire de l'instruction `return`. Il est nécessaire dans ce cas d'employer des techniques plus avancées (voir le chapitre 7, « Les classes et les objets »).

Fonction sans résultat

Une fonction peut ne pas fournir de résultat. Tel est, en général, le cas des fonctions utilisées pour l'affichage de messages. Par exemple, la fonction `menu()` suivante ne fournit pas de résultat et ne fait qu'exécuter les opérations selon la valeur du paramètre `choix` :

```
public static void menu (int choix)
{
  switch (choix)
  {
   case 1 :
    // Saisie d'une personne
   break;
   case 2 :
    // Afficher une personne
```

Copyright © 2016 Eyrolles.

```
    break;
   }
} // fin de menu()
```

L'en-tête `public static void menu (int choix)` mentionne que la fonction menu() ne retourne pas de résultat grâce au mot-clé `void` placé devant le nom de la fonction.

Si une fonction ne retourne pas de résultat, son en-tête est de type `void`, et l'instruction `return` ne figure pas dans le corps de la fonction.

Fonction à plusieurs paramètres

Prenons pour exemple une fonction `max()` qui fournisse en résultat la plus grande des deux valeurs données en paramètres :

```
import java.util.*;
public class Maximum  // Le fichier s'appelle Maximum.java
{
  public static void main(String [] parametre)
  {
   // Déclaration des variables
   int v1, v2, sup;
   Scanner lectureClavier = new Scanner(System.in);
   System.out.print("Entrer une valeur : ");
   v1 = lectureClavier.nextInt();
   System.out.print("Entrer une valeur : ");
   v2 = lectureClavier.nextInt();
   sup = max (v1,v2);
   System.out.print("le max de " + v1 + " et de " + v2 + " est " + sup);
} // fin de main()

  public static int max (int a, int b)
  {
   int m = a;
   if (b > m) m = b;
   return m;
  } // fin de max()
} //fin de class Maximum
```

La fonction max() possède un en-tête :

```
public static int max (int a, int b)
```

qui mentionne deux **paramètres**, `a` et `b`, de type `entier`.

Copyright © 2016 Eyrolles.

Nous observons que :

- Lorsqu'une fonction possède plusieurs paramètres, ceux-ci sont séparés par une virgule. L'en-tête d'une fonction peut alors prendre la forme suivante :

```
public static int quelconque (int a, char c, double t)
```

- Devant chaque paramètre est placé son type, même si deux paramètres consécutifs sont de type identique.

Question Que se passe-t-il si l'en-tête de la fonction `max()` est écrit de la façon suivante :

```
public static int max (int a, b)
```

Réponse Le compilateur affiche le message d'erreur suivant : `Identifier expected.`

Fonction sans paramètre

Une fonction peut ne pas avoir de paramètre. Son en-tête ne possède alors aucun paramètre entre parenthèses.

Ainsi, la fonction `sortie()` suivante permet de sortir proprement de n'importe quel programme :

```
public static void sortie ()
{
 System.out.print("Au revoir et a bientot...");
// Fonction Java qui permet de sortir proprement d'un programme
 System.exit(0);
}
```

Copyright © 2016 Eyrolles.

Résumé

Un algorithme paramétré est une marche à suivre qui fournit un résultat pouvant différer suivant la valeur du ou des paramètres. Dans le langage Java, les algorithmes paramétrés s'appellent des fonctions ou encore des méthodes.

Le langage Java propose un ensemble de fonctions prédéfinies fort utiles. Parmi ces fonctions, se trouvent les fonctions mathématiques, telles que `Math.sqrt()`, pour calculer la racine carrée du nombre placé entre `parenthèses`, ou `Math.log()`, pour le logarithme.

L'étude des fonctions mathématiques montre que :

- Pour exécuter une fonction, il est nécessaire d'écrire dans une instruction le nom de la fonction choisie, suivi des paramètres éventuels, placés entre parenthèses.
- Toute fonction possède, dans son nom d'appel, des parenthèses, ouvrante et fermante.
- Le type et l'ordre des paramètres dans l'appel de la fonction doivent être respectés, sous peine d'obtenir une erreur de compilation ou d'exécution.

Le langage Java offre en outre au programmeur la possibilité d'écrire ses propres fonctions, de façon à obtenir des programmes bien adaptés au problème qu'il doit résoudre. La définition d'une fonction passe par plusieurs étapes, qui permettent de :

- Préciser les instructions composant la fonction, en les plaçant dans le corps de la fonction. Ce dernier est déterminé par des accolades `{ }`.
- Associer le nom de la fonction aux instructions à l'aide d'un en-tête, qui précise le nom de la fonction, le type des paramètres (appelés paramètres formels) et le type de résultat retourné. Cet en-tête se rédige sous la forme suivante :

```
public static type nomdelafonction (paramètres)
```

- Établir les `paramètres` utiles à l'exécution de la fonction en les déclarant à l'intérieur des parenthèses placées juste après le nom de la fonction.
  - Lorsqu'une fonction possède plusieurs paramètres, ceux-ci sont séparés par une virgule. Devant chaque paramètre est placé son type, même si deux paramètres consécutifs sont de type identique.
  - Lorsqu'une fonction n'a pas de paramètre, son en-tête ne possède aucun paramètre entre parenthèses.
- Préciser le type de résultat fourni par la fonction dans l'en-tête de la fonction et placer l'instruction `return` dès que le résultat doit être transmis au programme appelant la fonction.
  - Toute fonction fournissant un résultat possède un `return` placé dans le corps de la fonction.
  - L'en-tête de la fonction possède obligatoirement un type, qui correspond au type de résultat retourné. Notons qu'une fonction ne retourne qu'une et une seule valeur.
  - Si une fonction ne retourne pas de résultat, son en-tête est de type `void`, et l'instruction `return` ne figure pas dans le corps de la fonction.

Une fonction peut être appelée (exécutée) depuis une autre fonction ou depuis la fonction `main()`, qui représente le programme principal. L'appel d'une fonction est réalisé en écrivant une instruction composée du nom de la fonction suivi, entre parenthèses, d'une liste de paramètres.

Copyright © 2016 Eyrolles.

Exercices

Apprendre à déterminer les paramètres d'un algorithme

Exercice 5.1 Pour écrire l'algorithme permettant de réaliser une boisson plus ou moins sucrée, procédez de la façon suivante :

a. Écrivez le bloc d'instructions qui place un nombre déterminé de morceaux de sucre dans une boisson chaude.

b. Déterminez le paramètre qui permet de sucrer plus ou moins la boisson.

c. Donnez un nom à l'algorithme et précisez le paramètre.

d. Écrivez l'algorithme en utilisant le nom du paramètre.

e. Appelez l'algorithme paramétré par son nom, en tenant compte du nombre de morceaux de sucre souhaité.

Comprendre l'utilisation des fonctions

Exercice 5.2 À la lecture du programme suivant :

```
public class Fonction
{
  public static void main(String [] parametre)
  {
   // Déclaration des variables
   int a,compteur;
   for (compteur = 0; compteur <= 5; compteur++)
   {
     a = calculer(compteur);
     System.out.print(a + " a ");
   }
  } // fin de main()

  public static int calculer(int x)
  {
    int y;
    y = x * x;
    return y ;
      } // fin de calculer()
    } //fin de class
```

Copyright © 2016 Eyrolles.

a. Délimitez les trois blocs définissant la fonction `main()`, la fonction `calculer()` et la classe `Fonction`.

b. Quel est le paramètre formel de la fonction `calculer()` ?

c. Quelles sont les valeurs transmises au paramètre de la fonction `calculer()` lors de son appel depuis la fonction `main()` ?

d. Quels sont les résultats produits par la fonction `calculer()` ?

e. Quelles sont les valeurs transmises à la variable `a` ?

f. Décrivez l'affichage réalisé par la fonction `main()`.

Exercice 5.3 Soit la fonction :

```
public static int f( int x)
{
 int resultat;
 resultat = -x * x + 3 * x - 2;
 return resultat;
}
```

a. Écrivez la fonction `main()` qui affiche le résultat de la fonction `f(x)` pour `x = 0`.

b. Transformez la fonction `main()` de façon à calculer et à afficher le résultat de la fonction pour `x` entier variant entre `-5` et `5`. Utilisez pour cela, dans la fonction `main()`, une boucle `for` avec un indice variant entre `-5` et `5`.

c. Pour déterminer le maximum de la fonction `f(x)` entre `-5` et `5`, calculez la valeur de `f(x)` pour chacune de ces valeurs, et stockez le maximum dans une variable `max`.

Détecter des erreurs de compilation concernant les paramètres ou le résultat d'une fonction

Exercice 5.4 Déterminez les erreurs de compilation des extraits de programmes suivants :

a. En utilisant la fonction `max()` décrite au cours de ce chapitre :

```
public static void main(String [] parametre)
{
 // Déclaration des variables
 double v1, v2, sup;
 Scanner lectureClavier = new Scanner(System.in);
 System.out.print("Entrer une valeur : ");
 v1 = lectureClavier.nextDouble();
 System.out.print("Entrer une valeur : ");
 v2 = lectureClavier.nextDouble();
 sup = max (v1,v2);
 System.out.print("Le max de " + v1 + " et " + v2 + " est " + sup);
} // fin de main()
```

Copyright © 2016 Eyrolles.

b.

```
public static int max (int a, int b)
{
  float m = a;
  if (m < b) m = b;
  return m;
} // fin de max()
```

c. En utilisant la fonction `menu()` décrite au cours de ce chapitre :

```
public static void main(String [] parametre)
{
// Déclaration des variables
   int v1, v2;
   Scanner lectureClavier = new Scanner(System.in);
   System.out.print("Entrer une valeur : ");
   v1 = lectureClavier.nextInt();
   v1 = menu (v2);
} // fin de main()
```

d.

```
public static void menu (int c)
{
   switch (c)
   {...
   }
  return c;
}
```

Écrire une fonction simple

Exercice 5.5 Écrivez la fonction `pourcentage()`, qui permet de calculer les pourcentages d'utilisation de la Carte Bleue, du chéquier et des virements automatiques, sachant que la formule de calcul du pourcentage pour la Carte Bleue est, comme nous l'avons vu au chapitre 1, « Stocker une information », la suivante :

Nombre de paiements par Carte Bleue / Nombre total de paiements * 100.

Suivez les étapes décrites dans le présent chapitre :

a. Déterminez les instructions composant la fonction.

b. Associez le nom de la fonction aux instructions.

c. Pour déterminer les paramètres de la fonction, recherchez les valeurs pouvant modifier le résultat du calcul.

Copyright © 2016 Eyrolles.

Remarque L'en-tête d'une fonction ayant deux paramètres entiers s'écrit :
`public static type nomdelafonction(int a, int b).`

d. Précisez le type de résultat fourni par la fonction.

e. Écrivez la fonction `main()` qui fait appel à la fonction `pourcentage()` et qui permette d'obtenir une exécution telle que :

```
Nombre de paiements par Carte Bleue : 5
Nombre de cheques emis : 10
Nombre de virements automatiques : 5
Vous avez emis 20 ordres de debit
dont 25.0 % par Carte Bleue
     50.0 % par cheque
     25.0 % par virement
```

Exercice 5.6 En vous inspirant de la structure de la fonction `f()` de l'exercice 5.3 et de la boucle `do...while` écrite au cours de l'exercice 4.2 de chapitre précédent :

a. Écrivez la fonction `vérifier()` qui demande la saisie d'une valeur tant que celle-ci est comprise entre `0` et `100`. Dès que la valeur saisie appartient à l'intervalle demandé, la fonction retourne la valeur saisie, en résultat.

b. Examinez le code suivant :

```
public class Exercice6  {
  public static void main(String [] parametre)  {
   int valeur;
   valeur = vérifier();
   System.out.print("valeur :" + valeur);
   }
  } // fin de main()
```

Que réalise l'application si l'utilisateur saisi les valeurs `-10`, `123` et `22` ?

c. Comment modifier la fonction `verifier()` pour que la valeur saisie soit comprise non plus entre `0` et `100` mais, entre deux valeurs `a` et `b` choisies par l'utilisateur.

d. Écrivez la fonction `verifierAvecBornes()` qui prend en compte cette modification et faites en sorte que la valeur saisie depuis la fonction `main()` soit comprise entre `10` et `20`.

Copyright © 2016 Eyrolles.

Le projet : Gestion d'un compte bancaire

Le programme écrit au chapitre 4, « Faire des répétitions », est suffisamment structuré pour y placer des fonctions. En effet, chaque option du projet est un programme à part entière et peut donc être décrite sous forme de fonction.

Dans le cadre de ce chapitre, nous allons construire trois fonctions relativement simples, qui vont nous permettre de comprendre le mécanisme de construction des fonctions.

Définir une fonction

Les fonctions sans paramètre avec résultat

La fonction `menuPrincipal()` affiche le menu principal du programme et demande la saisie de l'option choisie. Cette valeur doit être communiquée à la fonction `main()` pour exécuter la structure `switch` qui suit cette fonction.

a. Décrivez l'en-tête de la fonction `menuPrincipal()`, en prenant soin de préciser le type correspondant à la valeur retournée.

b. Placez les instructions relatives à l'affichage du menu et à la saisie de l'option dans le corps de la fonction.

c. Vérifiez que l'opérateur `return` soit appliqué à la variable contenant le choix de l'option.

Les fonctions sans paramètre ni résultat

La fonction `sortir()` affiche un message de politesse avant de sortir proprement du programme. Elle ne fournit pas de résultat et n'a pas non plus besoin de paramètre, puisque aucune valeur spécifique n'est nécessaire à son exécution.

a. Décrivez l'en-tête de la fonction `sortir()`.

b. Déterminez les instructions composant cette fonction et placez-les dans le corps de la fonction.

La fonction `alAide()` affiche à l'écran une explication sur ce que réalise chaque option de l'application.

a. Décrivez l'en-tête de la fonction `alAide()`.

b. Déterminez les instructions composant cette fonction et placez-les dans le corps de la fonction.

Appeler une fonction

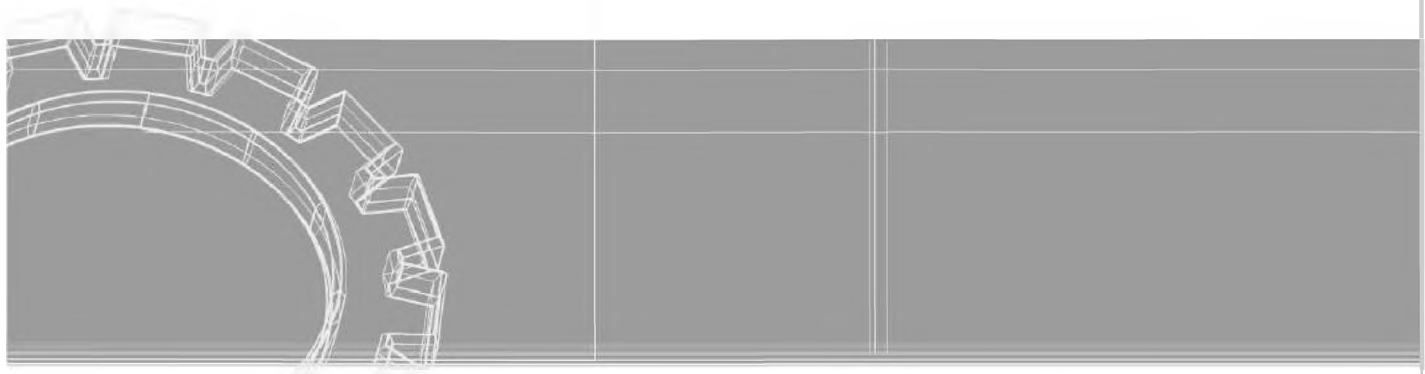
Modifiez la fonction `main()` de votre programme de façon à utiliser les trois fonctions `alAide()`, `sortir()` et `menuPrincipal()`, définies aux étapes précédentes.

L'exécution finale du programme doit être identique à celle du chapitre précédent. Seule la structure interne du programme est modifiée, ce dernier étant composé de quatre « blocs fonctions ».

Copyright © 2016 Eyrolles.

Chapitre 6

Fonctions, notions avancées



La création et l'utilisation de fonctions dédiées à la résolution d'un problème donné sont, nous l'avons observé au chapitre précédent, des opérations fondamentales, qui permettent le développement de logiciels dont le code source est facilement réutilisable.

Ces fonctions transforment la structure générale des programmes et apportent, de ce fait, de nouveaux concepts, qu'il est important de bien maîtriser avant d'étudier la programmation objet.

Nous commençons par examiner (section « La structure d'un programme »), ces nouvelles notions, telles que la visibilité des variables, les variables locales et les variables de classe, à partir d'exemples simples. Pour chacune de ces notions, nous observons leur répercussion sur le résultat des différents programmes donnés en exemple.

Nous analysons ensuite (section « Les fonctions communiquent »), comment les fonctions échangent des données par l'intermédiaire des paramètres et du retour de résultat. À partir de cette analyse, nous constatons que ces modes de communication ne permettent pas toujours d'obtenir l'opération souhaitée.

La structure d'un programme

Nous avons déjà observé (voir, au chapitre précédent, la section « Les fonctions au sein d'un programme Java ») qu'un programme était constitué d'une classe, qui englobe un ensemble de fonctions définissant chacune un bloc d'instructions indépendant.

Copyright © 2016 Eyrolles.

En réalité, il existe trois principes fondamentaux qui régissent la structure d'un programme Java. Ces principes sont détaillés ci-dessous.

1. Un programme contient :
   - une fonction principale, appelée fonction `main()` ;
   - un ensemble de fonctions définies par le programmeur ;
   - des instructions de déclaration de variables.
2. Les fonctions contiennent :
   - des instructions de déclaration de variables ;
   - des instructions élémentaires (affectation, test, répétition, etc.) ;
   - des appels à des fonctions, prédéfinies ou non.
3. Chaque fonction est comparable à une boîte noire, dont le contenu n'est pas visible en dehors de la fonction.

De ces trois propriétés, découlent les notions de visibilité des variables, de variables locales et de variables de classe. Concrètement, ces trois notions sont attachées au lieu de déclaration des variables, comme l'illustre la figure 6-1.

```
public class NomDeLaClasse
{
  //Déclaration de variables

  public static void main(String [] arg)
  {
   //Déclaration de variables

   //Instructions élémentaires (if, for,...)
   //Appel de fonctions prédéfinies ou non
  }

  public static type nomFonction(paramètre)
  {
    //Déclaration de variables

    //Instructions élémentaires (if, for,...)
    //Appel de fonctions prédéfinies ou non
  }
}
```

Figure 6-1 *Les variables peuvent être déclarées à l'intérieur ou à l'extérieur des fonctions mais toujours dans une classe.*

Pour mieux comprendre ces différents concepts, nous allons observer un programme composé de deux fonctions, `main()` et `modifier()`, et d'une variable, nommée `valeur`. La fonction `modifier()` a pour objectif de modifier le contenu de la variable `valeur`.

Copyright © 2016 Eyrolles.

Pour chaque exemple, la variable `valeur` est déclarée en un lieu différent du programme. À partir de ces variations, le programme fournit un résultat différent, que nous analysons.

Question Que se passe-t-il si l'on place l'instruction :

```
System.out.print("Bonjour ! ") ;
```

en dehors de toute fonction ?

Réponse Lors de la compilation, deux messages d'erreur s'affichent :

```
<identifier> expected System.out.print("Bonjour ! ") ;
cannot resolve symbol : class out
```

En effet, les seules instructions autorisées en dehors des fonctions sont les instructions de déclarations de variables.

La visibilité des variables

Après étude des trois propriétés énoncées ci-dessus, nous observons qu'un programme est constitué de **déclarations de variables** et de **fonctions**. Il existe, de fait, une notion d'extérieur et d'intérieur aux fonctions. Les instructions élémentaires, de type affectation, test, etc., se situent toujours à l'intérieur d'une fonction, alors que la déclaration de variables est une opération réalisable à l'intérieur ou à l'extérieur d'une fonction.

De plus, la troisième propriété énumérée ci-dessus exprime qu'une fonction ne peut pas utiliser dans ses instructions une variable déclarée dans une autre fonction. Pour mieux visualiser cette propriété, examinons le programme ci-dessous.

Exemple : code source complet

```
public class Visibilite
{
  public static void main(String [] arg)
   {
   // Déclaration des variables
   int valeur = 2 ;
   System.out.println("Valeur = " + valeur + " avant modifier() ");
   modifier();
   System.out.println("Valeur = " + valeur + " apres modifier() ");
   } // fin de main()
```

Copyright © 2016 Eyrolles.

```
  public static void modifier ()
  {
   valeur = 3 ;
   System.out.println("Valeur = " + valeur + " dans modifier() ");
  } // fin de modifier
} //fin de class Visibilite
```

Dans ce programme, nous constatons que l'instruction `valeur = 3 ;`, placée dans la fonction `modifier()`, cherche à modifier le contenu de la variable `valeur`, déclarée non pas dans la fonction `modifier()` mais dans la fonction `main()`.

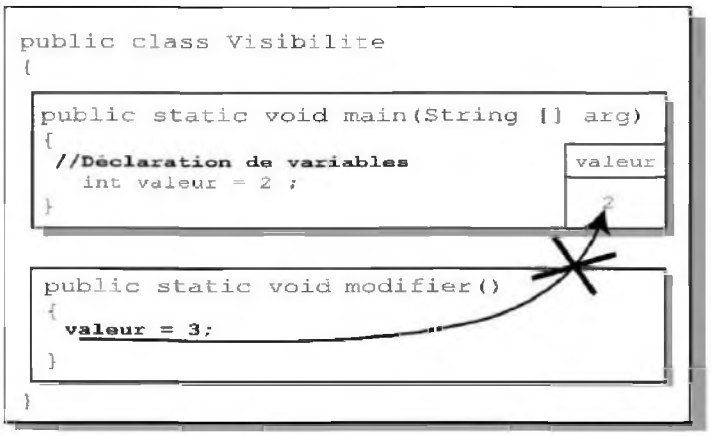


Figure 6-2 Une variable déclarée dans une fonction ne peut pas être utilisée par une autre fonction.

Cette modification n'est pas réalisable, car la variable `valeur` n'est définie qu'à l'intérieur de la fonction `main()`. Elle est donc **invisible** depuis la fonction `modifier()`. Les fonctions sont, par définition, des blocs distincts. La fonction `modifier()` ne peut agir sur la variable `valeur`, qui n'est visible qu'à l'intérieur de la fonction `main()`.

C'est pourquoi le fait d'écrire l'instruction `valeur = 3 ;` dans la fonction `modifier()` provoque une erreur de compilation du type : `Line 12 : Undefined variable : valeur`.

Variable locale à une fonction

La deuxième propriété énoncée précédemment établit qu'une fonction est formée d'instructions élémentaires, et notamment des instructions de déclaration de variables.

Par définition, une variable déclarée à l'intérieur d'une fonction est dite **variable locale à la fonction**. Pour l'exemple précédent, la variable `valeur` est locale à la fonction `main()`.

Copyright © 2016 Eyrolles.

Les variables locales n'existent que pendant le temps de l'exécution de la fonction. Elles ne sont pas modifiables depuis une autre fonction. Nous l'avons vu à la section précédente, le contenu de la variable `valeur` ne peut être modifié par une instruction située en dehors de la fonction `main()`.

Cependant, le programmeur débutant qui souhaite modifier à tout prix la variable `valeur` va chercher à corriger, dans un premier temps, l'erreur de compilation énoncée ci-dessus. Pour cela, il déclare une variable `valeur` à l'intérieur de la fonction `modifier()` et une autre à l'intérieur de la fonction `main()`. De cette façon, la variable `valeur` est définie dans chacune des fonctions, et aucune erreur de compilation n'est détectée. Examinons plus précisément ce que réalise un tel programme.

Exemple : code source complet

```
public class VariableLocale
{
  public static void main(String [] arg)
  {
   // déclaration de variables locales
   int valeur = 2 ;
   System.out.println("Valeur = " + valeur + " avant modifier() ");
   modifier();
   System.out.println("Valeur = " + valeur + " apres modifier() ");
  } // fin de main()

  public static void modifier ()
  {
   // déclaration de variables locales
   int valeur ;
   valeur = 3 ;
   System.out.println("Valeur = " + valeur + " dans modifier() ");
  } // fin de modifier
} //fin de class VariableLocale
```

Pour bien comprendre ce qu'effectue ce programme, construisons le tableau d'évolution de chaque variable déclarée dans le programme `VariableLocale.java`.

Pour en savoir plus Le tableau d'évolution des variables est décrit au chapitre 1, « Stocker une information », à la section « L'instruction d'affectation ».

Puisque les fonctions `main()` et `modifier()` sont des blocs d'instructions séparés, l'interpréteur Java crée un emplacement mémoire pour chaque déclaration de la variable `valeur`. Il existe deux cases mémoire `valeur` distinctes portant le même nom.

Copyright © 2016 Eyrolles.

Elles sont distinctes parce qu'elles n'appartiennent pas à la même fonction. Le tableau des variables déclarées pour chaque fonction est le suivant :

Variable locale à `main()`	valeur
`valeur = 2 ;`	2

Variable locale à `modifier()`	valeur
`valeur = 3 ;`	3

Résultat de l'exécution

L'exécution du programme a pour résultat :

```
Valeur = 2 avant modifier()
Valeur = 3 dans modifier()
Valeur = 2 apres modifier()
```

À l'exécution du programme, le premier appel à la fonction `System.out.println()` affiche le contenu de la variable `valeur` définie dans la fonction `main()`, soit 2.

Le programme réalise ensuite les actions suivantes :

- Appeler la fonction `modifier()`, qui affiche le contenu de la variable `valeur` définie à l'intérieur de cette fonction, soit 3.
- Sortir de la fonction `modifier()` et détruire la variable `valeur` locale à cette fonction.
- Retourner à la fonction `main()` et afficher de nouveau le contenu de la variable `valeur` définie dans la fonction `main()`, soit 2.

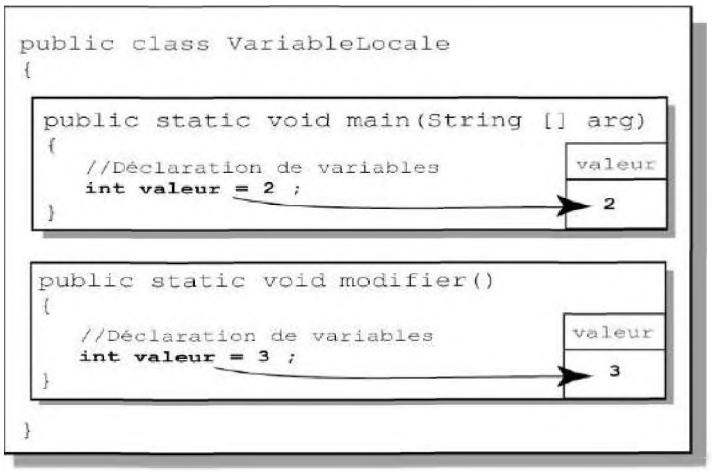


Figure 6-3 *Toute variable déclarée à l'intérieur d'une fonction est une variable locale, propre à cette fonction.*

Copyright © 2016 Eyrolles.

La variable `valeur` est déclarée deux fois dans chacune des deux fonctions, et nous constatons que la fonction `modifier()` ne change pas le contenu de la variable `valeur` déclarée dans la fonction `main()`. En réalité, même si ces deux variables portent le même nom, elles sont totalement différentes, et leur valeur est stockée dans deux cases mémoire distinctes.

En cherchant à résoudre une erreur de compilation, nous n'avons pas écrit la fonction qui modifie la valeur d'une variable définie en dehors d'elle-même. Cette modification est impossible dans la mesure où la variable `valeur` n'est connue que de la fonction, et d'aucune autre.

Variable de classe

En examinant plus attentivement la première propriété définie au début de ce chapitre (voir section « La structure d'un programme »), nous constatons que les classes contiennent également des instructions de déclaration, en dehors de toute fonction. Les variables ainsi déclarées sont appelées **variables de classe**. Elles sont définies pour l'ensemble du programme et sont visibles depuis toutes les fonctions.

La déclaration des variables de classe se réalise comme décrit ci-dessous.

Exemple : code source complet

```
public class VariableDeClasse
{
  // déclaration de variables de classe
  static int valeur ;
  public static void main(String [] parametre)
  {
   valeur = 2 ;
   System.out.println("Valeur = " + valeur + " avant modifier() ");
   modifier();
   System.out.println("Valeur = " + valeur + " apres modifier() ");
  } // fin de main()

  public static void modifier ()
  {
   valeur = 3 ;
   System.out.println("Valeur = " + valeur + " dans modifier() ");
  } // fin de modifier
} //fin de class VariableDeClasse
```

Copyright © 2016 Eyrolles.

Grâce à l'instruction `static int valeur ;`, la variable `valeur` est définie pour tout le programme `VariableDeClasse`. Le mot-clé `static` est important, car lorsque l'interpréteur Java le rencontre, il créé une case mémoire en un seul exemplaire, accessible depuis n'importe quelle méthode.

Pour en savoir plus Les propriétés du mot-clé `static` sont définies au chapitre 8, « Les principes du concept Objet », la section « La communication objet ».

La représentation par blocs du programme (voir figure 6-4) montre que la variable `valeur` est visible tout au long du programme.

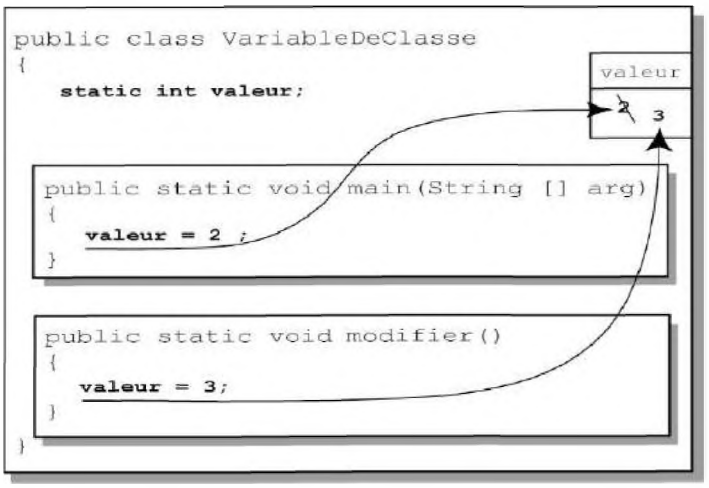


Figure 6-4 *Une variable déclarée en dehors de toute fonction est appelée variable de classe.*

Puisque la variable `valeur` est déclarée à l'extérieur des fonctions `main()` et `modifier()`, elle est définie comme étant une variable de la classe `VariableDeClasse`. La variable `valeur` existe tout le temps de l'exécution du programme, et les fonctions définies à l'intérieur de la classe peuvent l'utiliser et modifier son contenu.

Résultat de l'exécution

L'exécution du programme a pour résultat :

```
Valeur = 2 avant modifier()
Valeur = 3 dans modifier()
Valeur = 3 apres modifier()
```

Copyright © 2016 Eyrolles.

La variable `valeur` étant une variable de classe, l'ordinateur ne créé qu'un seul emplacement mémoire. Le tableau d'évolution de la variable est le suivant :

Variable de classe	valeur
`valeur = 2 // dans la fonction main()`	2
`valeur = 3 // dans la fonction modifier()`	3
`valeur = 3 // dans la fonction main()`	3

Puisqu'il n'existe qu'une seule case mémoire nommée `valeur`, celle-ci est commune à toutes les fonctions du programme, qui peuvent y déposer une valeur. Lorsque la fonction `modifier()` place `3` dans la case mémoire `valeur`, elle écrase la valeur `2`, que la fonction `main()` avait précédemment placée.

En utilisant le concept de variable de classe, nous pouvons écrire une fonction qui modifie le contenu d'une variable définie en dehors de la fonction.

Quelques précisions sur les variables de classe

Puisque les variables locales ne sont pas modifiables depuis d'autres fonctions et que, à l'inverse, les variables de classe sont vues depuis toutes les fonctions du programme, le programmeur débutant aura tendance, pour se simplifier la vie, à n'utiliser que des variables de classe.

Or, l'utilisation abusive de ce type de variables comporte plusieurs inconvénients, que nous détaillons ci-dessous.

Déclarer plusieurs variables portant le même nom

L'emploi systématique des variables de classe peut être source d'erreurs, surtout lorsqu'on prend l'habitude de déclarer des variables portant le même nom. Observons le programme suivant :

```
public class MemeNom
{
  // déclaration de variables de classe
  static int valeur ;
  public static void main(String [] parametre)
  {
   valeur = 2 ;
   System.out.println("Valeur = " + valeur + " avant modifier() ");
   modifier();
```

Copyright © 2016 Eyrolles.

```
    System.out.println("Valeur = " + valeur + " apres modifier() ");
  } // fin de main()

  public static void modifier ()
  {
   System.out.println(valeur + " dans modifier() avant la declaration");
   // Déclaration de variables locales
   int valeur ;
   valeur = 3 ;
   System.out.println(valeur + " dans modifier() apres la declaration");
  } // fin de modifier
} //fin de class MemeNom
```

Dans ce programme, la variable `valeur` est déclarée deux fois, une fois comme variable de classe et une autre fois comme variable locale à la fonction `modifier()`.

Remarque Rien n'interdit de déclarer plusieurs fois une variable portant le même nom dans des blocs d'instructions différents.

Le fait de déclarer deux fois la même variable n'est cependant pas sans conséquence sur le résultat du programme.

Dans la fonction `modifier()`, les deux variables `valeur` coexistent et représentent deux cases mémoire distinctes. Lorsque l'instruction `valeur = 3` est exécutée, l'interpréteur Java ne peut placer la valeur numérique 3 dans les deux cases mémoire à la fois. Il est obligé de choisir. Dans un tel cas, la règle veut que ce soit la variable locale qui soit prise en compte et non la variable de classe.

Le résultat final du programme est le suivant :

```
Valeur = 2 avant modifier()
2 dans modifier() avant la déclaration
3 dans modifier() après la déclaration
Valeur = 2 après modifier()
```

La modification n'est valable que localement. Lorsque le programme retourne à la fonction `main()`, la variable locale n'existe plus. Le programme affiche le contenu de la variable de classe, soit 2.

Le véritable nom d'une variable de classe

Une variable de classe se différencie des variables locales par son nom. Lorsqu'une variable de classe est déclarée, l'ordinateur lui donne un nom, qui lui permet de la distinguer des autres variables.

Copyright © 2016 Eyrolles.

Ce nom est constitué du nom de la classe, suivi d'un point puis du nom de la variable déclarée. Pour l'exemple suivant, la variable de classe `valeur` s'appelle en fait `VeritableNom.valeur`. Le programme peut s'écrire de la façon suivante :

```
public class VeritableNom
{
  // Déclaration de variables de classe
  static int valeur ;
  public static void main(String [] paramètre)
  {
   VeritableNom.valeur = 2 ;
   System.out.println(VeritableNom.valeur + " avant modifier() ");
   modifier();
   System.out.println(VeritableNom.valeur + " apres modifier() ");
  } // fin de main()

  public static void modifier ()
  {
   System.out.println("Variable de classe : " + VeritableNom.valeur );
   // Déclaration de variables locales
   int valeur = 3 ;
   System.out.println("Variable locale : " + valeur );
   VeritableNom.valeur = 3 ;
   System.out.println("Variable de classe : " + VeritableNom.valeur );
  } // fin de modifier
} //fin de class VeritableNom
```

En écrivant la variable de classe par son nom véritable, l'ambiguïté sur l'emploi de la variable de classe ou de la variable locale est levée, et l'exécution du programme a le résultat suivant :

```
2 avant modifier()
Variable de classe : 2
Variable locale : 3
Variable de classe : 3
3 après modifier()
```

Remarque

Pour éviter toute méprise, il est recommandé d'utiliser les variables de classe avec parcimonie et chaque fois avec leur nom complet. En pratique, seules les variables qui présentent un intérêt général pour le programme sont à déclarer comme variables de classe.

Copyright © 2016 Eyrolles.

De l'indépendance des fonctions

Comme nous l'avons déjà observé (voir, au chapitre précédent, la section « Algorithme paramétré »), une fonction est avant tout un sous-programme indépendant, capable d'être exécuté autant de fois que nécessaire et traitant des données différentes.

En construisant des fonctions qui utilisent des variables de classe, nous créons des fonctions qui ne sont plus des modules de programmes indépendants mais des extraits de programmes travaillant tous sur le même jeu de variables.

Cette dépendance aux variables de classe nuit au programme, car il est nécessaire, pour réutiliser de telles fonctions, de modifier tous les noms des variables de classe de façon à les rendre compatibles avec les nouveaux programmes.

En cas de développement de logiciels importants, comportant des centaines de milliers d'instructions, la transformation et l'amélioration des fonctionnalités du programme se trouvent fortement compromises. L'ensemble du code doit être examiné précisément afin de déterminer où se trouve la variable de classe concernée par la transformation envisagée.

Dans ce cadre, il convient de prendre les règles suivantes :

- Utiliser les variables de classe en nombre limité, le choix de ce type de variable s'effectuant en fonction de l'importance de la variable dans le programme. Une variable est considérée comme une variable de classe lorsqu'elle est commune à un grand nombre de fonctions.
- Écrire un programme de façon modulaire, chaque fonction travaillant de façon indépendante, à partir de valeurs transmises à l'aide des techniques étudiées à la section suivante.

Les fonctions communiquent

L'emploi systématique des variables de classe peut être, comme nous venons de le voir, source d'erreurs. Pour limiter leur utilisation, il existe des techniques simples, qui font que deux fonctions communiquent le contenu d'une case mémoire locale de l'une des fonctions à une case mémoire locale de l'autre.

Ces techniques sont basées sur le paramétrage des fonctions et sur le retour de résultat.

Pour en savoir plus Voir, au chapitre 5, « De l'algorithme paramétré à l'écriture de fonctions », la section « Les différentes formes d'une fonction ».

Pour mieux cerner le fonctionnement de chacune de ces techniques, nous allons les étudier à l'aide d'un programme composé de deux fonctions, `main()` et `tripler()`, et d'une variable `valeur`, locale à la fonction `main()`. La fonction `tripler()` a pour objectif de multiplier par trois le contenu de la variable `valeur`.

Copyright © 2016 Eyrolles.

Le passage de paramètres par valeur

Notre contrainte est cette fois de n'utiliser que des variables locales. Nous supposons donc que la variable `valeur` soit locale à la fonction `main()`. Pour multiplier par trois cette valeur, la fonction `tripler()` doit connaître effectivement le contenu de la variable `valeur`.

La fonction `main()` doit communiquer pour cela le contenu de la variable `valeur` à la fonction `tripler()`. Cette communication est réalisée en passant le contenu de la variable au paramètre de la fonction `tripler()`. Examinons le programme ci-dessous.

Exemple : code source complet

```
public class ParValeur
{
  public static void main (String [] arg)
  {
   // Déclaration des variables
   int valeur = 2 ;
   System.out.println("Valeur = " + valeur + " avant tripler() ");
   tripler(valeur);
   System.out.println("Valeur = " + valeur + " apres tripler() ");
  } // fin de main()

  public static void tripler (int valeur)
  {
   System.out.println("Valeur = " + valeur + " dans tripler() ");
   valeur = 3 * valeur;
   System.out.println("Valeur = " + valeur + " dans tripler() ");
  } // fin de tripler
} //fin de class ParValeur
```

Dans ce programme, deux variables `valeurs` sont déclarées. La première est locale à la fonction `main()`, tandis que la seconde est locale à la fonction `tripler()`. Cependant, comme la seconde est déclarée dans l'en-tête de la fonction, elle est considérée comme variable locale à la fonction et surtout, comme paramètre formel de la fonction `tripler()`.

Les paramètres formels sont définis au chapitre 5, « De l'algorithme paramétré à l'écriture de fonctions », la section « Définir une fonction ».

Copyright © 2016 Eyrolles.

De cette façon, lorsque la fonction `tripler()` est appelée depuis la fonction `main()`, avec comme valeur de paramètre, le contenu de `valeur` soit 2, la variable `valeur` locale de `tripler()` prend la valeur 2 (voir figure 6-5).

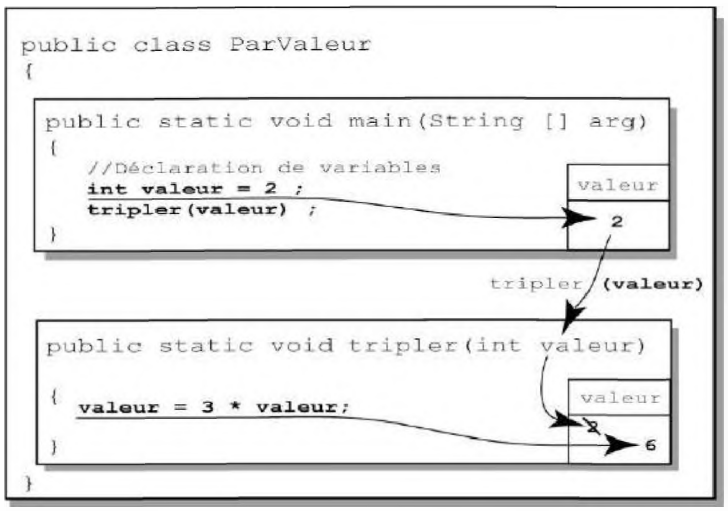


Figure 6-5 *Grâce au paramètre, le contenu d'une variable locale à la fonction appelante, main(), est transmis à la fonction appelée, tripler().*

Ensuite, la variable `valeur` locale à la fonction `tripler()` est multipliée par trois grâce à l'instruction `valeur = 3 * valeur;`. La variable `valeur` vaut donc 6 dans la fonction `tripler()`. Lorsque le programme sort de la fonction `tripler()` et retourne à la fonction `main()`, il détruit la variable locale de la fonction `tripler()` et affiche le contenu de la variable `valeur` locale à la fonction `main()`, soit encore 2.

Résultat de l'exécution

```
Valeur = 2 avant tripler()
Valeur = 2 dans tripler()
Valeur = 6 dans tripler()
Valeur = 2 apres tripler()
```

Grâce au paramètre de la fonction `tripler()`, le contenu de la variable `valeur` locale à la fonction `main()` est transmis à la fonction `tripler()`. Une fois la fonction exécutée, nous constatons que la variable `valeur` de la fonction `main()` n'est pas modifiée pour autant.

En effet, la valeur passée en paramètre est copiée dans la case mémoire associée au paramètre. Même si le paramètre porte le même nom que la variable, il s'agit de deux cases mémoire distinctes. La modification reste donc locale à la fonction.

Copyright © 2016 Eyrolles.

Remarque Lorsqu'une fonction communique le contenu d'une variable à une autre fonction par l'intermédiaire d'un paramètre, on dit que le **paramètre est passé par valeur.** Ce type de transmission de données ne permet pas de modifier, dans la fonction appelante, le contenu de la variable passée en paramètre.

Le résultat d'une fonction

Pour garder le résultat de la modification du contenu d'une variable en sortie de fonction, une technique consiste à retourner la valeur calculée par l'intermédiaire de l'instruction `return`.

Examinons le programme ci-dessous, qui utilise cette technique.

Exemple : code source complet

```
public class Resultat
{
  public static void main (String [] arg)
  {
   // Déclaration des variables
   int valeur = 2 ;
   System.out.println("Valeur = " + valeur + " avant tripler() ");
   valeur = tripler(valeur);
   System.out.println("Valeur = " + valeur + " apres tripler() ");
  } // fin de main()

  public static int tripler (int v)
  {
   System.out.println("v = " + v + " dans tripler() ");
   int résultat = 3*v ;
   System.out.println("Resultat = " + résultat
                                    + " dans tripler() ");
   return résultat;
  } // fin de tripler
} //fin de class Resultat
```

Ici, le contenu de la variable `valeur` est passé au paramètre `v` de la fonction `tripler()`. Puisque le paramètre formel (`v`) correspond à une case mémoire distincte de la variable effectivement passée (`valeur`), il est plus judicieux de le déclarer sous un autre nom d'appel que

Copyright © 2016 Eyrolles.

celui de la variable, de façon à ne pas les confondre. En général, et tant que cela reste possible, nous avons pour convention de donner comme nom d'appel du paramètre formel la première lettre du paramètre réel. Pour notre exemple, `valeur` est le paramètre réel. Le paramètre formel s'appelle donc `v`.

Une fois le calcul réalisé à l'intérieur de la fonction `tripler()`, la valeur résultante placée dans la variable `résultat` est transmise à la fonction `main()` qui a appelé la fonction `tripler()`. Cette transmission est réalisée grâce à l'instruction `return résultat;`. Le contenu du résultat est alors placé dans la variable `valeur` grâce au signe d'affectation `=`, comme l'illustre la figure 6-6.

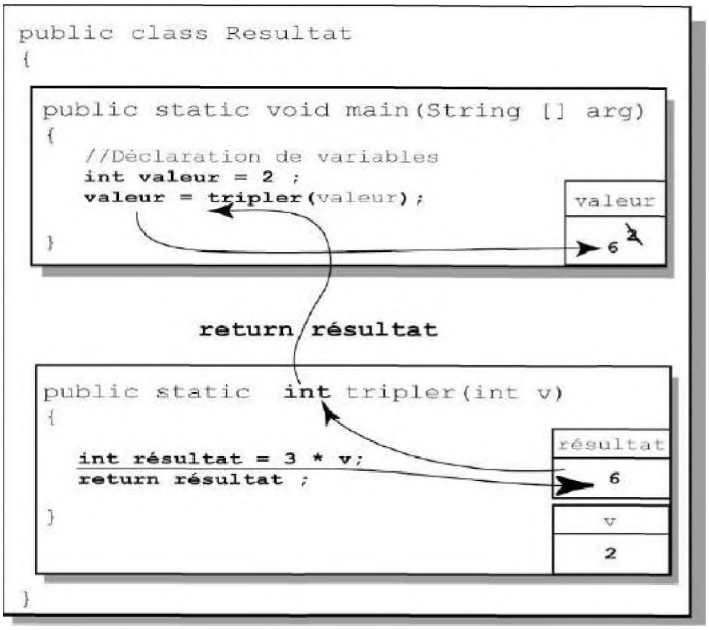


Figure 6-6 *Grâce au retour de résultat, le contenu d'une variable locale à la fonction appelée tripler() est transmis à la fonction appelante main().*

Résultat de l'exécution

```
Valeur = 2 avant tripler()
v = 2 dans tripler()
Resultat = 6 dans tripler()
Valeur = 6 après tripler()
```

Grâce à la technique du retour de résultat et du passage de paramètre par valeur, les fonctions peuvent échanger les contenus de variables. Les variables locales sont donc exploitables aussi facilement que les variables de classe, tout en évitant les inconvénients liés à ces dernières.

Copyright © 2016 Eyrolles.

Lorsqu'il y a plusieurs résultats à retourner

Une difficulté subsiste : le retour de résultat ne peut se réaliser que sur une seule valeur. Il n'est pas possible de retourner plusieurs valeurs à la fois. Si l'on souhaite écrire l'algorithme qui échange le contenu de deux variables sous forme de fonction, nous nous trouvons confronté au problème décrit dans l'exemple ci-dessous.

Pour en savoir plus Sur l'échange de valeurs reportez-vous, au chapitre 1, « Stocker une information », à la section « Échanger les valeurs de deux variables ».

Exemple : code source complet

```
import java.util.*;
public class PlusieursResultats
{
  public static void main (String [] arg)
  {
    int a, b;
    Scanner lectureClavier = new Scanner(System.in);
    System.out.print("Entrer une valeur pour a : ");
    a = lectureClavier.nextInt();
    System.out.print("Entrer une valeur pour b : ");
    b = lectureClavier.nextInt();
    System.out.println(" a = "+a+" b = "+b);
    échange (a,b);
    System.out.print("Apres echange, ");
    System.out.println("a = "+a+" b = "+b);
  }

  public static void échange(int x, int  y)
  {
    int tmp = x;
    x = y;
    y = tmp;
  }
}
```

La fonction `échange()` réalise théoriquement l'échange du contenu des deux variables passées en paramètres. Si `a` prend la valeur `1` et que `b` vaut `2`, après exécution de la fonction `échange()` `a` doit prendre la valeur de `b`, soit `2`, et `b` la valeur de `a`, soit `1`.

Copyright © 2016 Eyrolles.

Question Quel est le résultat du programme `PlusieursResultats`, en supposant que l'utilisateur entre au clavier les valeurs `1` puis `2` ?

Réponse

```
Entrer une valeur pour a : 1
Entrer une valeur pour b : 2
a = 1 b = 2
Apres echange,
a = 1 b = 2
```

En effet, les valeurs de `a` et `b` sont copiées dans les paramètres `x` et `y`. L'échange des valeurs n'est donc réalisé que pour les variables `x` et `y` locales à la fonction `échange()`. Lorsque la fonction `échange()` a fini d'être exécutée, `a` et `b` n'ont pas été modifiées.

Nous le constatons à l'exécution : aucun échange n'a été réalisé. Il n'y a rien d'étonnant à cela, puisque le passage des paramètres est un passage par valeur et qu'il ne modifie pas le contenu des paramètres réels `a` et `b` passés à la fonction `échange()`.

La solution qui consiste à retourner le résultat est impossible. En effet, il serait nécessaire de retourner les deux variables échangées, et il n'est pas possible d'écrire `return x, y ;`, la syntaxe de cette instruction n'étant pas valide

Pour en savoir plus Voir, au chapitre 5, « De l'algorithme paramétré à l'écriture de fonctions », la section « Les différentes formes d'une fonction ».

Dans l'état actuel de nos connaissances, nous ne sommes pas à même de récupérer différentes valeurs modifiées au sein d'une fonction. En réalité, seul le concept d'objet permet de réaliser un tel exploit. Nous l'étudions au chapitre suivant.

Copyright © 2016 Eyrolles.

Résumé

Un programme Java est structuré selon les trois principes fondamentaux suivants :

- Un programme contient :
  - une fonction principale, appelée fonction `main()` ;
  - un ensemble de fonctions définies par le programmeur ;
  - des instructions de déclaration de variables.
- Les fonctions contiennent :
  - des instructions de déclaration de variables ;
  - des instructions élémentaires (affectation, test, répétition, etc.) ;
  - des appels à des fonctions, prédéfinies ou non.

  Chaque fonction est comparable à une boîte noire, dont le contenu n'est pas visible en dehors de la fonction.

De ces trois propriétés découlent les notions suivantes :

- **Visibilité** : toute variable déclarée à l'intérieur d'une fonction n'est visible que dans cette fonction et ne peut être utilisée dans une autre fonction.
- **Variable locale** : toute variable déclarée à l'intérieur d'une fonction est une variable locale à cette fonction. Ces variables n'existent que le temps de l'exécution de la fonction, et elles ne sont pas modifiables depuis une autre fonction.
- **Variable de classe** : les variables déclarées en dehors de toute fonction sont appelées des variables de classe. Ces variables sont définies pour l'ensemble du programme, et elles sont visibles et modifiables par toutes les fonctions de la classe.

Lorsqu'une variable de classe et une variable locale portant le même nom coexistent à l'intérieur d'une fonction, la règle veut que ce soit la variable locale qui soit prise en compte et non la variable de classe.

Les fonctions sont des blocs d'instructions distinctes. Pour communiquer le contenu d'une case mémoire (variable) locale de l'une à une case mémoire locale de l'autre fonction, il est nécessaire d'utiliser les techniques suivantes :

- **Les paramètres des fonctions** : lorsqu'une fonction communique le contenu d'une variable à une autre fonction par l'intermédiaire d'un paramètre, on dit que le paramètre est passé **par valeur**. Ce type de transmission de données ne permet pas de modifier, dans la fonction appelante, le contenu de la variable passée en paramètre.
- **Le retour de résultat** : pour garder en résultat la modification du contenu d'une variable en sortie de fonction, une technique consiste à retourner la valeur calculée par l'intermédiaire de l'instruction `return`.

Ces deux modes de communication ne permettent pas de récupérer plusieurs données modifiées à l'intérieur d'une fonction. Seul le concept d'objet, étudié au chapitre suivant, permet de réaliser cette opération.

Copyright © 2016 Eyrolles.

Exercices

Repérer les variables locales et les variables de classe

Exercice 6.1 En observant le programme suivant :

```
import java.util.*;
public class Calculette
{
 public static double résultat ;

 public static void main( String [] argument)
 {
  int a, b;
  Scanner lectureClavier = new Scanner(System.in);
  menu();
  System.out.println("Entrer la premiere valeur ");
  a = lectureClavier.nextInt();
  System.out.println("Entrer la seconde valeur ");
  b = lectureClavier.nextInt();
  calculer();
  afficher();
 }

 public static void calculer()
 {
  char opération ;
  switch (opération)
  {
   case '+' : résultat = a + b ;
              break ;
   case '-' : résultat = a - b ;
              break ;
   case '/' : résultat = a / b ;
              break ;
   case '*' : résultat = a * b ;
              break ;
  }
 }
```

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

```
public static void afficher()
{
 char opération ;
 System.out.print(a + "  " +opération + "  "+ b + " =  " + résultat);
}

public static void menu()
{
 char opération ;
 Scanner lectureClavier = new Scanner(System.in);
 System.out.println("Je sais compter, entrez l'operation choisie") ;
 System.out.println(" + pour additionner ") ;
 System.out.println(" - pour soustraire ") ;
 System.out.println(" * pour multiplier ") ;
 System.out.println(" / pour diviser ") ;
 System.out.println(" (+, -, *, /)  ? : ") ;
 opération = lectureClavier.next().charAt(0);
}
}
```

a. Recherchez les différentes fonctions définies dans la classe `Calculette`.

b. Dessinez le programme sous forme de schéma, en représentant les fonctions à l'aide de blocs. Placez les variables dans les blocs où elles sont déclarées.

c. À l'aide du schéma, déterminez les variables locales à chacune des fonctions, ainsi que les variables de classe.

d. Après exécution de la fonction `menu()` et lecture des deux valeurs numériques `a` et `b`, la fonction `calculer()` peut-elle réaliser l'opération demandée ? Pourquoi ?

e. Même question pour la fonction `afficher()`.

Communiquer des valeurs à l'appel d'une fonction

Exercice 6.2 Pour corriger le programme `Calculette`, nous supposons que les variables `résultat` et `opération` sont déclarées en tant que variables de classe et non plus localement aux fonctions `afficher()` et `menu()`.

a. Modifiez le schéma réalisé en 1.b, en tenant compte de ces nouvelles déclarations.

b. Quelle technique doit-on utiliser pour que les fonctions `calculer()` et `afficher()` connaissent le contenu des variables `a` et `b`, afin d'effectuer ensuite les instructions qui les composent ?

c. Écrivez les fonctions en utilisant cette technique.

Copyright © 2016 Eyrolles.

Transmettre un résultat à la fonction appelante

Exercice 6.3 Nous supposons que le programme `Calculette` ne contienne plus de variables de classe. Les variables `résultat` et `opération` sont maintenant déclarées localement aux fonctions qui les utilisent.

a. Quelles sont les conséquences de cette nouvelle hypothèse sur le résultat du programme ?

b. Comment la fonction `calculer()` peut-elle connaître l'opérateur choisi par l'utilisateur dans la fonction `menu()` ?

c. Transformez la fonction `menu()` de sorte que l'opérateur soit transmis à la fonction `main()`.

d. Modifiez la fonction `calculer()` de façon à lui transmettre l'opérateur fourni par la fonction `menu()`.

e. Comment la fonction `afficher()` peut-elle connaître le résultat de la fonction `calculer()` ?

f. Transformez la fonction `calculer()` de sorte que le résultat soit transmis à la fonction `main()`.

g. Modifiez la fonction `afficher()` de façon à lui transmettre le résultat fourni par la fonction `calculer()`.

Le projet : Gestion d'un compte bancaire

Au chapitre précédent, nous avons construit trois fonctions, `alAide()`, `sortir()` et `menuPrincipal()`, qui améliorent la lisibilité du programme. Ces fonctions concernent surtout l'affichage de messages de dialogue de l'application vers l'utilisateur (menu, aide, etc.). Elles réalisent l'interface entre l'utilisateur et l'application sans transformer les données propres à chaque compte bancaire.

Pour réaliser les opérations de création et d'affichage d'un compte (options `1` et `2` du menu), nous allons ici construire des fonctions qui modifient, transforment les données d'un compte.

Comprendre la visibilité des variables

La fonction `afficherCpte()` réalise l'option `2` du menu principal de notre application. Cette fonction affiche l'ensemble des caractéristiques d'un compte, soit son numéro, son type, son taux, s'il s'agit d'un compte d'épargne, et sa valeur courante. Nous supposons, que l'ensemble de ces valeurs aient été préalablement saisies en option `1`.

Les variables locales

Une première solution pourrait s'écrire :

```
public static void afficherCpte()
{
  long num ;
  char type ;
  double taux ;
```

Copyright © 2016 Eyrolles.

```
double val ;
System.out.print("Le compte n° : " + num + " est un compte ");
if (type == 'C') System.out.println(" courant  ");
else if (type == 'J') System.out.println(" joint  ");
else if (type == 'E')
{
  // affiche le taux dans le cas d'un compte d'épargne.
  System.out.println(" epargne  dont le taux est  " + taux);
}
System.out.println(" Valeur initiale : " + val);
}
```

Quelles valeurs sont affichées par cette fonction ? Pourquoi ?

Les variables de classe

Pour corriger la fonction précédente, il est nécessaire que la fonction ait accès aux valeurs stockées lors de l'option 1.

Une première solution consiste à définir les variables à afficher comme variables de classe.

a. Transformez votre programme, et déclarez les variables `num`, `type`, `taux` et `val` comme variables de classe.

b. Retirez les déclarations des variables `num`, `type`, `taux` et `val` dans la fonction `afficherCpte()` de façon à éviter qu'elles soient encore utilisées par l'interpréteur comme variables locales.

c. Exécutez votre programme et vérifiez que la fonction affiche correctement les valeurs.

Le passage de paramètres par valeur

Une seconde solution revient à déclarer les variables `num`, `type`, `taux` et `val` en paramètres de la fonction d'affichage, de façon à transmettre les valeurs saisies depuis la fonction `main()` (option 1) à la fonction `afficherCpte()`.

a. Décrivez l'en-tête de la fonction `afficherCpte()`, en prenant soin de déclarer en paramètre une variable pour chaque caractéristique du compte à transmettre à la fonction.

b. Déterminez les instructions composant cette fonction, et placez-les dans le corps de la fonction.

Les limites du retour de résultat

La fonction `créerCpte()` rassemble les instructions de l'option 1, soit l'affichage de messages et la saisie au clavier des valeurs caractéristiques d'un compte.

a. Recherchez quel doit être le résultat de la fonction à transmettre à la fonction `main()`.

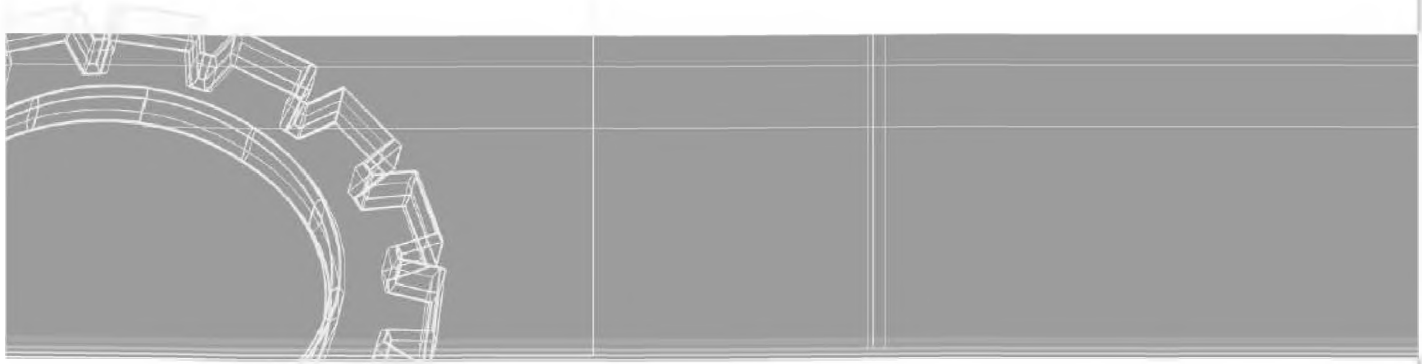
b. Pour décrire l'en-tête de la fonction `créerCpte()`, est-il possible de déterminer le type à placer dans l'en-tête de la fonction ? Pourquoi ?

Copyright © 2016 Eyrolles.

Copyright © 2016 Eyrolles.

Chapitre 7

Les classes et les objets



L'étude du chapitre 6, « Fonctions, notions avancées », montre que, si une fonction fournit plusieurs résultats, ceux-ci ne peuvent pas être transmis au programme appelant. Pour contourner cette difficulté, il est nécessaire d'utiliser des objets, au sens de la programmation objet.

Pour comprendre les principes fondamentaux de la notion d'objet, nous étudions (section « La classe String, une approche vers la notion d'objet »), comment définir et gérer des objets de type `String`. Ce type permet la représentation des mots en tant que suites de caractères. À partir de cette étude, nous analysons les instructions qui font appel aux objets `String` afin d'en comprendre les principes de notation et d'utilisation.

Nous examinons ensuite (section « Construire et utiliser ses propres classes ») comment définir de nouveaux types de données. Pour cela, nous déterminons les caractéristiques syntaxiques d'une classe et observons comment manipuler des objets à l'intérieur d'une application et comment utiliser les méthodes qui leur sont associées.

La classe String, une approche de la notion d'objet

La classe `String` est une classe prédéfinie du langage Java. Elle permet de définir des « variables » contenant des suites de caractères, autrement dit des mots, ou, dans le jargon informatique, des **chaînes de caractères**. Nous étudions comment définir ces « variables » à la section ci-après.

Copyright © 2016 Eyrolles.

La classe `String` est un type de données composé d'un grand nombre d'outils, ou méthodes, qui facilitent l'utilisation des chaînes de caractères (voir la section « Les différentes méthodes de la classe `String` »).

Manipuler des mots en programmation

L'utilisation des chaînes de caractères apporte beaucoup à la convivialité des programmes informatiques. Il serait impensable aujourd'hui de créer un logiciel de gestion du personnel sans pouvoir définir le nom et le prénom de chaque employé. Dans le même ordre d'idée, que serait la recherche d'informations sur Internet sans ces fameuses chaînes de caractères ?

Grâce aux chaînes de caractères, nous oublions le langage binaire, et il devient aisé de communiquer avec l'ordinateur dans notre propre langue. Pourtant, l'utilisation de ces fameuses chaînes a longtemps été source de difficultés.

Les mots nécessitent un type de données particulier, du fait qu'un mot possède par nature, un nombre quelconque de caractères. À la différence des formats `int`, `double` ou `char`, les chaînes de caractères ne peuvent *a priori*, être représentées par un nombre fixe de cases mémoire.

Déclaration d'une chaîne de caractères

Tout comme nous déclarons des variables pour stocker des valeurs entières ou réelles, nous devons déclarer une variable pour mémoriser la suite des caractères d'un mot ou d'une phrase. Le type de cette variable est le type `String`.

Le type `String` n'est pas un type simple, puisqu'il permet de regrouper sous un seul nom de variable plusieurs données, c'est-à-dire l'ensemble des caractères d'un mot.

Pour éviter les difficultés liées à la variation du nombre de caractères dans un mot, le langage Java fixe la longueur du mot en fonction de sa déclaration. Cela fait, le contenu du mot ne peut plus être modifié. En déclarant un objet de type `String`, il est possible, en même temps, de l'initialiser en lui affectant des caractères placés entre guillemets.

La déclaration suivante permet de créer un objet appelé `mot`, qui contient la chaîne de caractères `"exemple"` :

```
String mot = "exemple";
```

Notons que la variable `mot` n'est pas un ensemble de sept cases mémoire contenant les sept caractères du mot `exemple`. Lors de la déclaration de la variable `mot`, l'interpréteur Java crée une case qui contient l'adresse de la case où se trouve le premier caractère du mot `exemple` (voir figure 7-1).

Lorsque l'ordinateur souhaite afficher la variable `mot`, il va rechercher l'information se situant à l'adresse stockée dans la case mémoire `mot`. On dit alors que la variable `mot` **pointe** sur la case qui contient la suite de caractères.

Copyright © 2016 Eyrolles.

Remarque Les variables de type `String` ne contiennent pas directement l'information qui les caractérise mais seulement l'adresse où trouver cette information. Dès lors, ces variables ne s'appellent plus des variables mais des objets.

Les objets, au sens de la programmation objet, ne sont pas des « variables » de type simple (`int`, `long`, `double`, `char`, etc.). Ils correspondent à un type qui permet de regrouper plusieurs données sous une même adresse.

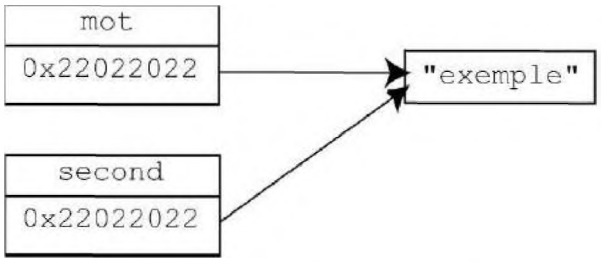


Figure 7-1 *Seul un objet de type String contenant le mot "exemple" existe. mot et second font tous deux référence à cet objet unique.*

Lorsqu'un objet `second` est déclaré comme ci-dessous, il contient la même adresse (référence) que l'objet `mot`.

```
String second = mot;
```

Si le programme modifie le contenu de l'objet mot en lui affectant par exemple, une nouvelle chaîne, l'interpréteur ne modifie pas la case pointée par `mot`, dans la mesure où par définition, le contenu d'un mot ne peut être modifié.

```
mot = "nouveau";
```

Il crée en réalité une nouvelle adresse et lui associe la nouvelle chaîne de caractères. Pour notre exemple, l'objet `mot` est associé à la chaîne de caractères `"nouveau"`, et `second` reste associé à `"exemple"`.

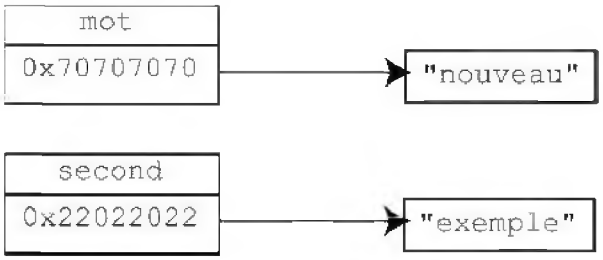


Figure 7-2 *La modification de mot entraîne la création d'une nouvelle chaîne de caractères et d'une nouvelle référence, automatiquement attribuées à mot. L'objet second conserve la précédente référence.*

Copyright © 2016 Eyrolles.

Les différentes méthodes de la classe String

L'utilisation des mots dans un programme est aujourd'hui incontournable. Il ne s'agit certes pas simplement d'afficher des mots mais de les traiter de la façon la plus intelligente possible. Ces traitements sont par exemple, le tri alphabétique ou encore la recherche de mots particuliers dans un texte.

Pour réaliser ces opérations, le langage Java propose un ensemble de méthodes prédéfinies.

Remarque

Les méthodes d'une classe sont comparables aux fonctions, mais la terminologie « objet » les appelle **méthodes**.

Ces méthodes offrent la possibilité de traiter rapidement et simplement l'information textuelle. Nous décrivons ci-dessous, regroupées par thème, une grande partie des méthodes définies dans la classe `String`. Nous donnons en exemple, pour chaque thème, un programme qui utilise ces méthodes.

Recherche de mots et de caractères

Opération	Fonction Java
Recherche si le mot se termine par le ou les caractères passés en paramètres.	`endsWith()`
Recherche si le mot commence par le ou les caractères passés en paramètres.	`startsWith()`
Recherche le caractère placé à la position spécifiée en paramètre. Le premier caractère occupe la position `0` et le dernier la position `length()-1` (voir ci-dessous la description de `length()`).	`charAt()`
Localise un caractère ou une sous-chaîne dans un mot, à partir du début du mot. Renvoie la valeur `-1` si le caractère ou la chaîne recherché ne fait pas partie du mot.	`indexOf()`
Localise un caractère ou une sous-chaîne dans un mot à partir de la fin du mot. Renvoie la valeur `-1` si le caractère ou la chaîne recherché ne fait pas partie du mot.	`lastIndexOf()`
Extrait une sous-chaîne d'un mot.	`substring()`

Copyright © 2016 Eyrolles.

Exemple de recherche de mots et de caractères

```
import java.util.*;
public class Rechercher {
 public static void main(String [] argument)
 {
  String phrase = "Mieux vaut tard que jamais";
  String soumo ="";
  int place;
  Scanner lectureClavier = new Scanner(System.in);
  System.out.println("Vous avez dit : " + phrase);
  soumo = phrase.substring(11,15);
  System.out.println("De 11 a 15, la sous chaine est : " + soumo);
  for ( int i = 0; i < 5; i++)
   System.out.println("en " + i + ", il y a : " + phrase.charAt(i));

  System.out.println("Entrez un mot : ");
  soumo = lectureClavier.next();

  if(phrase.endsWith(soumo))
   System.out.println("La phrase se termine avec : " + soumo);
  else
   System.out.println("La phrase ne finit pas avec : " + soumo);

  place = phrase.indexOf(soumo);
  if (place == -1)
   System.out.println("Ce mot n'existe pas dans : " + phrase);
  else
   System.out.println(soumo+" est a la position " + place);
 }
}
```

Copyright © 2016 Eyrolles.

Question Quel est le résultat de l'exécution du programme `Recherche` si l'utilisateur saisit le mot `tard` au clavier ?

Réponse

```
Vous avez dit : Mieux vaut tard que jamais
De 11 a 15, la sous-chaîne est : tard
En 0, il y a : M
En 1, il y a : i
En 2, il y a : e
En 3, il y a : u
En 4, il y a : x
Entrez un mot :  tard
La phrase ne finit pas avec :tard
tard est a la position : 11
```

`Phrase` et `soumo` sont deux objets de type `String`, initialisés respectivement à `"Mieux vaut tard que jamais"` et `""` (mot ne comportant pas de caractère).

L'instruction `soumo = phrase.substring(11,15);` recherche la sous-chaîne située entre les caractères `11` et `15` de l'objet `phrase`. Cela fait, elle place l'ensemble de ces caractères dans l'objet `soumo`.

Grâce à l'instruction `phrase.charAt(i)`, placée dans l'instruction d'affichage `System.out.print`, le programme affiche les cinq premiers caractères de l'objet `phrase`, `i` variant de `0` à `4`.

Ensuite, `phrase.endsWith(soumo)` permet de savoir si l'objet `phrase` se termine avec la suite de caractères saisie au clavier et stockée dans l'objet `soumo`. Le résultat de la méthode `endsWith()` est `true` si la chaîne se termine par l'argument et `false` dans le cas contraire. Pour notre exemple, `soumo` vaut `"tard"`, et la méthode retourne `false`. Le programme exécute donc l'instruction placée dans le bloc `else` associé au test `if(phrase.endsWith(soumo))`.

Pour finir, l'instruction `phrase.indexOf(soumo);` recherche si l'objet `soumo` est contenu dans l'objet `phrase`. Si tel est le cas, elle retourne la position du premier caractère trouvé, sinon elle retourne `-1`. Ici, `tard` est détecté dans `"mieux vaut tard que jamais"` en position 11.

Copyright © 2016 Eyrolles.

Question Quel est le résultat de l'exécution du programme `Recherche` si l'utilisateur saisi le mot `mais` au clavier ?.

Réponse

```
Vous avez dit : Mieux vaut tard que jamais
De 11 a 15, la sous-chaîne est : tard
En 0, il y a : M
En 1, il y a : i
En 2, il y a : e
En 3, il y a : u
En 4, il y a : x
Entrez un mot : mais
La phrase se termine avec : mais
mais est a la position : 22
```

Si l'utilisateur saisit `"mais"` au lieu de `"tard"`, le test `if (phrase.endsWith(soumo))` est vrai, la méthode `endsWith()` retournant `true`. Le programme exécute donc l'instruction placée dans le bloc `if`.

Question Quel est le résultat de l'exécution du programme `Recherche` si l'utilisateur saisi le mot `OK` au clavier ?

Réponse

```
Vous avez dit : Mieux vaut tard que jamais
De 11 a 15, la sous-chaîne est : tard
En 0, il y a : M
En 1, il y a : i
En 2, il y a : e
En 3, il y a : u
En 4, il y a : x
Entrez un mot : OK
La phrase ne finit pas avec : OK
Ce mot n'existe pas dans : Mieux vaut tard que jamais
```

Si l'utilisateur saisit `"OK"` au lieu de `"mais"`, le test `if (phrase.endsWith(soumo))` est faux, et la méthode `endsWith()` retourne `false`. Le programme exécute l'instruction placée dans le bloc `else`. De plus, l'instruction `phrase.indexOf(soumo);` retourne `-1` car `"OK"` n'est pas détecté dans `"mieux vaut tard que jamais"`. Le programme exécute alors le bloc `if` associé.

Copyright © 2016 Eyrolles.

Comparaison de mots

Opération	Fonction Java
Compare deux mots et retourne une valeur : - Nulle si les deux mots sont identiques. - Positive si le premier mot est plus grand (placé après) le deuxième mot (dans le dictionnaire). - Négative si le premier mot est plus petit (placé avant) le deuxième mot (dans le dictionnaire).	`compareTo()`
Compare la valeur de deux mots. Elle retourne `true` si les deux chaînes sont identiques et `false` dans le cas contraire.	`equals()`
Compare la valeur de deux mots sans différencier les majuscules des minuscules. Elle retourne `true` si les deux chaînes sont identiques et `false` dans le cas contraire.	`equalsIgnoreCase()`
Détermine si deux portions de chaînes sont identiques. Dans l'affirmative, elle renvoie `true`.	`regionMatches()`

Exemple de comparaison de mots

```
public class Comparer
 {
 public static void main(String [] argument)
 {
  String prvb1 = "Le mieux est l'ennemi du bien";
  String prvb2 ="Le Mieux Est l'Ennemi du bien";

  System.out.println("1 : " + prvb1);
  System.out.println("2 : " + prvb2);

  System.out.println("Comparons les 10 premiers caracteres : ");

  System.out.print("En tenant compte des majuscules : ");
  if (prvb1.regionMatches(false, 0, prvb2, 0, 10))
   System.out.println("Les 10 premiers cars sont identiques");
  else
   System.out.println("Il y a des differences sur les 10 premiers
                       cars");

  System.out.print("Sans tenir compte des majuscules : ");
  if (prvb1.regionMatches(18, prvb2, 18, 6))
   System.out.println("Les cars de 18 a 23 sont identiques");
  else
   System.out.println("Il y a des differences");

  if (prvb1.compareTo(prvb2) == 0)
```

Copyright © 2016 Eyrolles.

```
    System.out.println("Les deux chaines sont identiques");
   else
   {
    if (prvb1.compareTo(prvb2) < 0)
      System.out.print(prvb1 + " est avant " + prvb2);
    else
      System.out.print(prvb1 + " est apres " + prvb2);
    System.out.println("dans le dictionnaire");
   }
   System.out.print("Sans tenir compte des majuscules : ");
   if (prvb1.equalsIgnoreCase(prvb2))
     System.out.println("Les deux chaines sont identiques");
    else
     System.out.println("Les deux chaines sont differentes");
  }
}
```

Résultat de l'exécution

```
1 : Le mieux est l'ennemi du bien
2 : Le Mieux Est l'Ennemi du bien
Comparons les 10 premiers caracteres :
En tenant compte des majuscules : Il y a des differences sur
les 10 premiers cars
Sans tenir compte des majuscules : Les cars de 18 a 23 sont
identiques
Le mieux est l'ennemi du bien est apres Le Mieux Est l'Ennemi
du bien dans le dictionnaire;
Sans tenir compte des majuscules : Les deux chaines sont iden-
tiques
Les objets prvb1 et pvrb2 sont initialisés respectivement à "Le
mieux est l'ennemi du bien" et "Le Mieux Est l'Ennemi du bien".
```

La méthode `regionMatches()` s'utilise soit avec quatre paramètres, soit avec cinq paramètres. Dans ce programme, nous donnons en exemple les deux appels possibles :

- Le premier appel à la méthode utilise cinq paramètres (`regionMatches(false, 0, prvb2, 0, 10)`). Le premier paramètre est un booléen qui, s'il est égal à `false`, permet de réaliser la comparaison des deux mots, en tenant compte de la présence des majuscules. Pour notre cas, la méthode détermine si les deux portions de chaîne `prvb1` et `pvrb2` (correspondant au troisième paramètre de la méthode) sont identiques, en tenant compte des majuscules.

  Cette recherche est réalisée à partir de la valeur spécifiée par le deuxième paramètre (soit `0`, c'est-à-dire le premier caractère de `prvb1`). Le quatrième paramètre représente la posi-

Copyright © 2016 Eyrolles.

tion du premier caractère à comparer dans l'objet `prvb2`. Le cinquième est le nombre de caractères consécutifs à comparer. Pour notre exemple, le programme recherche s'il y a des similitudes entre `prvb1` et `prvb2`, à partir du début des deux mots, et ce sur dix caractères.

- Le deuxième appel à la méthode est composé de quatre paramètres (`regionMatches (18, prvb2, 18, 6)`). En fait, ces quatre paramètres correspondent aux quatre derniers paramètres de l'appel décrit précédemment. Le booléen figurant dans l'appel précédant n'existe plus, car, par défaut, cette méthode travaille sans tenir compte des majuscules. Elle est donc équivalente à l'appel de la méthode suivante : `prvb1.regionMatches (true, 18, prvb2, 18, 6)`.

Ensuite, l'instruction `prvb1.compareTo(prvb2)` compare les objets `prvb1` et `prvb2` et détermine s'ils sont identiques ou placés avant ou après dans l'ordre alphabétique.

Pour finir, l'instruction `prvb1.equalsIgnoreCase(prvb2)` vérifie si les deux objets `prvb1` et `prvb2` sont identiques ou non, sans tenir compte de la présence des majuscules.

Exemple de comparaison de mots avec switch

```
class ComparerAvecSwitch {

 public static void main(String [] argument) {
   String quelleCouleur = "";
   Scanner lectureClavier = new Scanner(System.in);
   System.out.println("Quelle couleur choisissez - vous (rouge,0
   ➥vert, orange, bleu, violet, jaune) ? : ") ;
   quelleCouleur = lectureClavier.nextLine();
   switch (quelleCouleur) {
    case  "vert" :
       System.out.println("Vous devez melanger du bleu avec du
       ➥jaune ") ;
    break ;
    case  "violet" :
       System.out.println("Vous devez melanger du bleu avec du
       ➥rouge ") ;
    break ;
    case  "orange" :
       System.out.println("Vous devez melanger du rouge avec du
       ➥jaune ") ;
    break ;
     default :
       System.out.println("C'est une couleur primaire !") ;
   } // Fin du switch
 }
}
```

Copyright © 2016 Eyrolles.

Grâce à la nouvelle structure `switch` de la version 7 de Java, il devient plus facile de comparer l'égalité de deux mots. Dans cet exemple, l'utilisateur saisit le nom d'une couleur et le programme affiche la composition de la couleur correspondante. La couleur choisie par l'utilisateur est stockée dans la variable `quelleCouleur`. Cette variable est ensuite comparée à chacune des couleurs proposées en choix, grâce à la structure `switch` et aux étiquettes `vert`, `violet`, `default`, etc. Selon la valeur de `quelleCouleur`, le programme affiche le mélange de couleurs correspondant.

Remarque

L'utilisateur doit saisir le nom de la couleur selon le format précisé dans la question, à savoir, tout en minuscule pour être sûr d'obtenir le bon résultat. Dans le cas contraire, le programme affiche que la couleur est une couleur primaire.

Transformation de formats

Opération	Fonction Java
Transforme en minuscules la chaîne sur laquelle la méthode est appliquée.	`toLowerCase()`
Transforme en majuscules la chaîne sur laquelle la méthode est appliquée.	`toUpperCase()`
La méthode place (concatène) la chaîne spécifiée en paramètre à la suite de la chaîne sur laquelle la méthode est appliquée.	`concat()`
Remplace systématiquement dans la chaîne sur laquelle la méthode est appliquée tous les caractères donnés en premier argument par le caractère donné en deuxième argument.	`replace()`
Calcule le nombre de caractères de la chaîne sur laquelle la méthode est appliquée.	`length()`

Exemple de transformation de format

```
public class Transformer
{
  public static void main(String [] argument)
  {
   String phrase = "Qui dort ";
   String verbe = "dine";
   String p1 = "", p2 = "", p3 = "", p4 = "";
   int nbcar;
   System.out.println("1 : " + phrase);
```

Copyright © 2016 Eyrolles.

```
    System.out.println("2 : " + verbe);
    p1 = phrase.toUpperCase();
    System.out.println("En majuscules : " + p1);
    p2 = phrase.toLowerCase();
    System.out.println("En minuscules : " + p2);
    p3 = phrase.concat(verbe);
    nbcar = p3.length();
    System.out.print("Apres concat() : ");
    System.out.println(p3 +" possede : " + nbcar + " caracteres");
    p4 = p3.replace('i','a');
    System.out.println("Apres replace() : " + p3 + " devient : " + p4);
  }
}
```

Résultat de l'exécution

```
1 : Qui dort
2 : dine
En majuscules : QUI DORT
En minuscules : qui dort
Apres concat() : Qui dort dine possede : 13 caracteres
Apres replace() : Qui dort dine devient : Qua dort dane
```

Les objets `phrase` et `verbe` sont initialisés respectivement à `"Qui dort "` et `"dine"`. L'instruction `p1 = phrase.toUpperCase();` transforme en majuscules le contenu de `phrase` et place cette transformation dans l'objet `p1`.

L'instruction `p2 = phrase.toLowerCase()` place dans `p2` le contenu de `phrase` transformé en minuscules. Notons que, pour chacune de ces instructions, l'objet `phrase` n'est jamais modifié.

L'instruction `p3 = phrase.concat(verbe);` place en bout de l'objet `phrase` le mot contenu dans `verbe`. Cela fait, le résultat de cette opération est affecté à l'objet `p3`. L'objet `phrase` n'est pas modifié.

Ensuite, l'instruction `nbcar = p3.length();` calcule la longueur de l'objet `p3`, c'est-à-dire le nombre de caractères constituant l'objet `p3`.

Pour finir, l'instruction `p4 = p3.replace('i', 'a');` remplace tous les caractères `'i'` de `p3` par des `'a'` et place le résultat de cette transformation dans l'objet `p4`. L'objet `p3` n'est pas modifié.

Copyright © 2016 Eyrolles.

Question Quel changement de format appliquer à `quelleCouleur` pour être sûr que le nom de la couleur saisie soit identique à celui des étiquettes définies dans la structure switch (voir la section précédente « Exemple de comparaison de mots avec switch ») ?

Réponse Pour améliorer la performance du programme et obtenir une réponse correcte, vous devez insérer, juste après la saisie de la couleur, l'instruction :

```
quelleCouleur= quelleCouleur.toLowerCase();
```

De cette façon, la variable `quelleCouleur` ne contient plus que des minuscules, même si l'utilisateur a saisi un nom de couleur avec des majuscules. La comparaison de `quelleCouleur` avec les étiquettes a donc plus de chances d'être valide pour ensuite obtenir un résultat correct.

Appliquer une méthode à un objet

L'observation des exemples précédents montre que l'appel d'une méthode de la classe `String` ne s'écrit pas comme une simple instruction d'appel à une méthode (fonction), telle que nous l'avons étudiée jusqu'à présent.

Comparons l'appel à une méthode de la classe `Math` à celui d'une méthode de la classe `String`.

Par exemple, pour calculer la valeur absolue d'une variable `x`, les instructions sont les suivantes :

```
double x = 4, y;
y = Math.abs(x) ;
```

Pour transformer un mot en lettres majuscules, les instructions sont :

```
String mot = "petit", MOT;
MOT =  mot.toUpperCase();
```

Comme nous le constatons, dans le premier cas, la fonction `Math.abs()` s'applique à la variable `x`, en passant la valeur de `x` en paramètre. En effet, les variables `x` et `y` ne sont pas des objets au sens de la programmation objet. Elles sont de type `double` et représentent simplement le nom d'une case mémoire dans laquelle l'information est stockée. Aucune méthode, aucun traitement ne sont associés à cette information.

Dans la seconde écriture, la méthode `toUpperCase()` est appliquée à l'objet `mot` par l'intermédiaire d'un point (.), placé entre le nom de l'objet et la méthode. Les objets `mot` et `MOT` ne peuvent être considérés comme des variables. Ils sont de type `String`. L'information représentée par ce type n'est pas simple. Elle représente (voir figure 7-3) les éléments suivants :

- D'une part, une référence (une adresse) vers un ensemble de caractères stockés dans plusieurs cases mémoire distinctes.
- D'autre part, un ensemble de méthodes propres qui lui sont applicables. Ces méthodes sont l'équivalent d'une boite à outils, qui opère uniquement sur les objets de type `String`.

Copyright © 2016 Eyrolles.

Question Les instructions suivantes sont-elles valides ?

```
int x = x.Math.sqrt() ;
String mot = toUpperCase(mot) ;
```

Réponse Aucune des deux instructions n'est valide. En effet, dans la première instruction la fonction `Math.sqrt()` est appliquée à `x`, qui n'est pas un objet, mais une variable de type `int`.

Dans la seconde instruction, la méthode `toUpperCase()` est employée comme une simple fonction alors qu'elle ne peut être appelée qu'à travers un objet de type `String`.

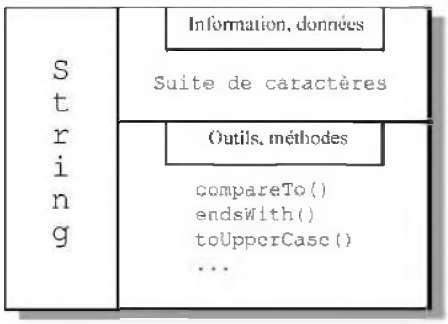


Figure 7-3 *La classe String définit l'association de données et de méthodes applicables à ces données.*

Remarque Quelle qu'elle soit, une classe correspond à un type, qui spécifie une association de données (informations ou valeurs de tout type) et de méthodes (outils d'accès et de transformation des données). Ces méthodes, définies à l'intérieur d'une classe, ne peuvent s'appliquer qu'aux données de cette même classe.

Grâce à cette association, une classe permet la définition de nouveaux types de données, qui structurent l'information à traiter (voir, dans ce chapitre, la section « Construire et utiliser ses propres classes »).

Principes de notation

À cause de cette différence fondamentale de représentation de l'information, l'emploi des méthodes à travers les objets utilise une syntaxe particulière.

Pour un objet de type `String`, cette syntaxe est la suivante :

```
// Déclaration et initialisation
String objet = "";
// La méthode s'applique à objet
objet.nomDeLaMéthode(liste des paramètres éventuels) ;
```

Copyright © 2016 Eyrolles.

Pour appliquer une méthode à un objet, il suffit de placer derrière le nom de l'objet un point suivi du nom de la méthode et de ses paramètres.

Remarque

Par convention :

- Tout nom de méthode commence par une minuscule.
- Si le nom de la méthode est composé de plusieurs mots, ceux-ci voient leur premier caractère passer en majuscule.
- Le nom d'une classe commence toujours par une majuscule.

Grâce à cette écriture, l'objet est associé à la méthode, de façon à pouvoir modifier l'information (les données) contenue dans l'objet. Cette technique permet de récupérer les différentes données modifiées localement par une méthode. Elle est le principe de base du concept d'objet, décrit et commenté au chapitre suivant.

Construire et utiliser ses propres classes

L'étude de la classe `String` montre qu'une classe correspond à un type de données. Ce type est composé de données et de méthodes exploitant ces données. La classe `String` est un type prédéfini du langage Java.

Il existe d'autres types prédéfinis (classes) dans le langage Java. Ces classes sont des outils précieux et efficaces, qui simplifient le développement des applications. Différentes classes sont examinées dans la troisième partie de cet ouvrage.

L'intérêt des classes réside aussi dans la possibilité de définir des types structurés, propres à un programme. Grâce à cette faculté, le programme se développe de façon plus sûre, les objets qu'il utilise étant définis en fonction du problème à résoudre.

Avant d'étudier réellement l'intérêt de la programmation objet et ses conséquences sur les modes de programmation (voir chapitre 8, « Les principes du concept d'objet »), nous examinons dans les sections qui suivent comment créer des types spécifiques et utiliser les objets associés à ces nouveaux types.

Définir une classe et un type

Définir une classe, c'est construire un type de données structuré. Avant de comprendre les avantages d'une telle construction, nous abordons ici la notion de type structuré (et donc de classe) d'un point de vue syntaxique.

Copyright © 2016 Eyrolles.

Pour définir un type, il suffit d'écrire une classe, qui, par définition, est constituée de données et de méthodes (voir figure 7-3). La construction d'une classe est réalisée selon les deux principes suivants :

1. **Définition des données** à l'aide d'instructions de déclaration de variables et/ou d'objets. Ces variables sont de type simple, tel que nous l'avons utilisé jusqu'à présent (`int`, `char`, etc.) ou de type composé, prédéfini ou non (`String`, etc.).

   Ces données décrivent les informations caractéristiques de l'objet que l'on souhaite définir. Elles sont aussi appelées communément champ, attribut ou membre de la classe.

2. **Construction des méthodes** définies par le programmeur. Ce sont les méthodes associées aux données. Elles se construisent comme de simples fonctions, composées d'un en-tête et d'instructions, comme nous l'avons vu aux chapitres précédents.

   Ces méthodes représentent tous les traitements et comportements de l'objet que l'on cherche à décrire.

En définissant de nouveaux types, nous déterminons les caractéristiques propres aux objets que l'on souhaite programmer. Un type d'objet correspond à l'ensemble des données traitées par le programme, regroupées par thème.

Un objet peut être une `personne`, si l'application à développer gère le personnel d'une société, ou un `livre`, s'il s'agit d'une application destinée à la gestion d'une bibliothèque. Signalons que l'objet `personne` peut aussi être utilisé dans le cadre d'un logiciel pour bibliothèque, puisqu'un lecteur empruntant un `livre` est aussi une `personne`.

Construire un type Cercle

Examinons, sur un exemple simple, la démarche de construction d'un type structuré. Observons pour cela comment construire le type de données qui décrit au mieux la représentation d'un cercle quelconque.

Cette réalisation passe par deux étapes : « Rechercher les caractéristiques propres à tout cercle » et « Définir le comportement de tout cercle ».

Rechercher les caractéristiques propres à tout cercle

D'une manière générale, tout cercle est défini grâce à son rayon. Si l'on souhaite afficher ce cercle, il est en outre nécessaire de connaître sa position à l'écran. Pour simplifier, nous supposons que la position d'un cercle est déterminée grâce aux coordonnées de son centre.

Les caractéristiques d'un cercle sont son rayon et sa position à l'écran, c'est-à-dire les coordonnées en `x` (abscisse) et en `y` (ordonnée) du centre du cercle. Ces trois données sont représentables à l'aide de valeurs numériques, que nous choisissons, pour simplifier, de type `int`.

Pour déclarer les données d'un cercle, nous écrivons les déclarations suivantes :

```
public int x, y; // position du centre du cercle
public int r ;   // rayon
```

Copyright © 2016 Eyrolles.

Définir le comportement de tout cercle

D'un point de vue informatique, plusieurs opérations peuvent être appliquées à un cercle. Un cercle peut être déplacé ou agrandi (voir les méthodes `déplacer()` et `agrandir()` dans le code source ci-après). Ces opérations modifient la valeur du rayon ou des coordonnées du centre du cercle à l'écran.

C'est pourquoi il est nécessaire de définir une méthode qui affiche à l'écran les données (rayon, position) d'un cercle avant ou après transformation (voir la méthode `afficher()` dans le code source ci-après).

La méthode de calcul du périmètre d'un cercle peut également être utile (voir la méthode `périmètre()` dans le code source ci-après).

La classe descriptive du type Cercle

```
public class Cercle
{
 public int x, y; // position du centre
 public int r;    // rayon

 //Affichage des données de la classe
 public void afficher()
 {
  System.out.println(" Cercle centre en " + x + ", " + y);
  System.out.println(" de rayon : " + r);
 }

 // Calcul du périmètre d'un cercle
 public double périmètre()
 {
  return 2*Math.PI*r;
 }

 // Déplace le centre du cercle en nx, ny). Ces coordonnées
 // étant passées en paramètres de la fonction
 public void déplacer(int nx, int ny)
 {
  x = nx;
  y = ny;
 }

// Augmente la valeur courante du rayon avec la valeur
// passée en paramètre
 public void agrandir(int nr)
 {
  r = r + nr;
 }
} // Fin de la classe Cercle
```

Copyright © 2016 Eyrolles.

La classe `Cercle`, décrite à l'intérieur d'un fichier appelé `Cercle.java`, définit un type de données composé de trois attributs caractéristiques des cercles, à savoir la position du centre en abscisse et ordonnée et le rayon, ainsi que quatre comportements différents. Sa description par bloc est représentée à la figure 7-4.

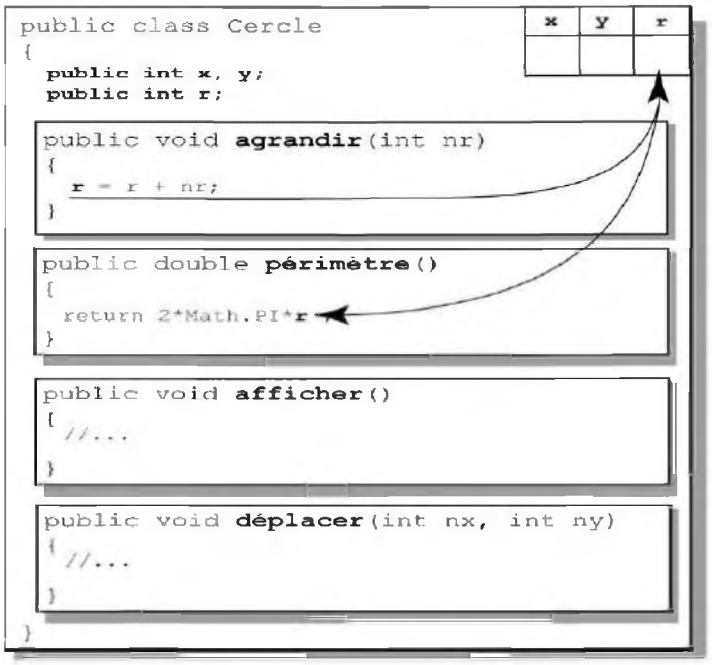


Figure 7-4 *Les données x, y et r du type Cercle sont déclarées en dehors de toute fonction. N'importe quelle modification de ces données est donc visible par l'ensemble des méthodes de la classe.*

Quelques observations

Suivant la description de la figure 7-4, nous constatons que les données `x`, `y` et `r` sont déclarées en dehors de toute fonction. Par conséquent, chaque méthode a accès à tout moment aux valeurs qu'elle contient, soit pour les consulter, soit pour les modifier.

Les méthodes `afficher()` et `périmètre()` ne font que consulter le contenu des données `x`, `y` et `r` pour les afficher ou les utiliser en vue d'obtenir un nouveau résultat.

Au contraire, les méthodes `déplacer()` et `agrandir()` modifient le contenu des données `x`, `y` et `r`. Ces modifications, réalisées à l'intérieur d'une méthode, sont aussi visibles depuis les autres méthodes de la classe.

Il existe donc deux types de méthodes, les méthodes qui permettent d'accéder aux données de la classe et celles qui modifient ces données.

Copyright © 2016 Eyrolles.

Pour en savoir plus Les différents types de méthodes sont décrits au chapitre 8, « Les principes du concept d'objet », la section « Les méthodes d'accès aux données ».

Remarques En comparant les programmes construits aux chapitres précédents à celui-ci, nous constatons les deux différences fondamentales suivantes :

- Le mot-clé `static` a disparu de toutes les instructions de déclaration. Cette disparition n'est pas sans conséquence sur le déroulement du programme. Elle permet de créer non plus de simples variables mais des objets (voir, au chapitre 8, « Les principes du concept d'objet », la section « Les données static »).

- Une classe définissant un type structuré ne possède pas de fonction `main()`. La définition d'une classe n'est pas la même chose que la réalisation d'une application. Une classe est une entité à part entière, qui définit globalement de quoi est constitué un objet et précise les opérations qu'il est possible de lui appliquer.

Bien entendu, une classe est définie pour être utilisée dans un programme exécutable (une application) qui contient une fonction `main()`. Nous abordons plus en détail cette opération à la section suivante.

Définir un objet

Après avoir défini un nouveau type structuré, l'étape suivante consiste à écrire une application qui utilise effectivement un « objet » de ce type. Pour cela, le programmeur doit déclarer les objets utiles à l'application et faire en sorte que l'espace mémoire nécessaire soit réservé.

Déclarer un objet

Cette opération simple s'écrit comme une instruction de déclaration, avec cette différence que le type de la variable n'est plus un type simple prédéfini mais un type structuré, tel que nous l'avons construit précédemment. Ainsi, dans :

```
// Déclaration d'un objet chose
TypeDeL'Objet chose ;
```

`TypeDeL'Objet` correspond à une classe définie par le programmeur. Pour notre exemple, la déclaration d'un cercle `A` est réalisée par l'instruction :

```
Cercle A ;
```

Cette déclaration crée une case mémoire, nommée `A`, destinée à contenir une référence vers l'adresse où sont stockées les informations concernant le cercle `A`. À ce stade, aucune adresse

Copyright © 2016 Eyrolles.

n'est encore déterminée.

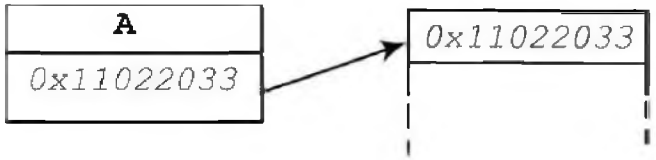


Figure 7-5 *La déclaration d'un objet réserve une case mémoire destinée à contenir l'adresse mémoire où seront stockées les informations. L'espace mémoire et l'adresse ne sont pas encore réservés pour réaliser ce stockage.*

Réserver l'espace mémoire à l'aide de l'opérateur new

À cette étape, les informations caractérisant l'objet `A` ne peuvent être stockées, car l'espace mémoire servant à ce stockage n'est pas encore réservé. C'est l'opérateur `new` qui réalise cette réservation.

L'opérateur `new` est un programme Java, qui gère de lui-même la réservation de l'espace mémoire. Lorsqu'on applique cet opérateur à un objet, il détermine combien d'octets lui sont nécessaires pour stocker l'information contenue dans la classe.

Cet opérateur s'applique en écrivant à la suite du terme `new` le nom du type de l'objet déclaré, suivi de deux parenthèses.

```
// Réserver de l'espace mémoire pour l'objet chose
chose = new TypeDeL'Objet();
```

Pour notre exemple, la réservation de l'espace mémoire pour définir le cercle `A` s'écrit :

```
A = new Cercle() ;
```

Notons qu'il est possible de déclarer et de réserver de l'espace mémoire en une seule instruction :

```
Cercle A = new Cercle() ;
```

En écrivant une telle instruction, nous observons que, pour chaque objet déclaré, l'opérateur `new` réserve suffisamment d'espace mémoire pour stocker les données de la classe et pour copier les méthodes associées. Il détermine aussi l'adresse où sera stocké l'ensemble de ces informations (l'espace mémoire pour l'objet `A` est illustré à la figure 7-6).

Lors de cette réservation, l'interpréteur initialise les données de la classe à `0` pour les entiers, à `0.0` pour les réels, à `'\0'` pour les `char` et à `null` pour les `String`. Pour notre exemple, `A` est un cercle de rayon nul centré en (0, 0).

Une instance est donc, en mémoire, un programme à part entière, composé de variables et de fonctions. Sa structure est telle qu'il ne peut s'exécuter et se transformer (c'est-à-dire modifier ses propres données) qu'à l'intérieur de cet espace. C'est pourquoi il est considéré comme une entité indépendante, ou « objet ».

Copyright © 2016 Eyrolles.

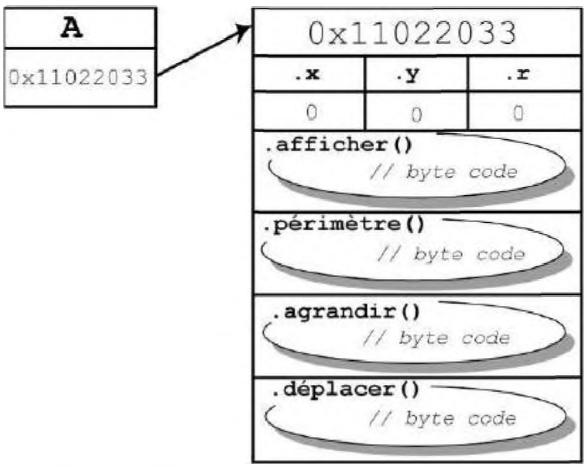


Figure 7-6 *Pour chaque objet créé, l'opérateur new réserve un espace mémoire suffisamment grand pour y stocker les données et les méthodes descriptives de la classe. L'adresse est alors déterminée.*

Remarque

L'objet ainsi défini est un représentant particulier de la classe, caractérisé par l'ensemble de ses données. Dans le jargon informatique, on dit que l'objet `A` est une **instance** de la classe `Cercle`. Les données qui le caractérisent, à savoir `x`, `y` et `r`, sont appelées des **variables d'instance**.

Manipuler un objet

L'objet ainsi défini est entièrement déterminé par ses données et ses méthodes. Il est dès lors possible de modifier les valeurs qui le caractérisent et d'exploiter ses méthodes.

Accéder aux données de la classe

Pour accéder à une donnée de la classe de façon à la modifier, il suffit d'écrire :

```
// Accéder à un membre de la classe
chose.nomDeLaDonnée = valeur du bon type ;
```

en supposant que le champ `nomDeLaDonnée` soit défini dans la classe correspondant au type de l'objet `chose`.

Pour notre exemple, la saisie au clavier des valeurs caractérisant le cercle `A` s'écrit de la façon suivante :

```
Scanner lectureClavier = new Scanner(System.in);
System.out.println(" Entrez la position en x : ") ;
A.x = lectureClavier.nextInt();
System.out.println(" Entrez la position en y : ") ;
A.y = lectureClavier.nextInt();
System.out.println(" Entrez le rayon : ");
A.r = lectureClavier.nextInt();
```

Copyright © 2016 Eyrolles.

Les cases mémoire représentant les variables d'instance (`x`, `y` et `r`) de l'objet `A` sont accessibles via l'opérateur `point (.)`.

Accéder aux méthodes de la classe

Pour appliquer une méthode de la classe à un objet particulier, la syntaxe utilise le même principe de notation :

```
// appliquer une méthode à l'objet chose
chose.nomDeLaMéthode(liste des paramètres éventuels) ;
```

en supposant que la méthode ait préalablement été définie pour le type de l'objet `chose`. Pour notre exemple, l'application de la méthode `périmètre()` à l'objet `A` s'écrit :

```
double p = A.périmètre();
```

Une application qui utilise des objets Cercle

L'exemple suivant montre comment exploiter dans une application, l'ensemble des données et des méthodes définies dans la classe `Cercle`.

Exemple : code source complet

```
import java.util.*;
public class FaireDesCercles
{
 public static void main(String [] arg)
 {
  Cercle A  = new Cercle();
  Scanner lectureClavier = new Scanner(System.in);
  A.afficher();
  System.out.println(" Entrez la position en x : ") ;
  A.x = lectureClavier.nextInt();
  System.out.println(" Entrez la position en y : ") ;
  A.y = lectureClavier.nextInt();
  System.out.println(" Entrez le rayon : ");
  A.r = lectureClavier.nextInt();
  A.afficher();

  double p = A.périmètre();
  System.out.println(" Votre cercle a pour perimetre : " + p);
  A.déplacer(5, 2);
  System.out.println(" Apres deplacement : ");
  A.afficher();
  A.agrandir(10);
  System.out.println(" Apres agrandissement : ");
  A.afficher();
 }
}
```

Copyright © 2016 Eyrolles.

Compilation et exécution d'une application multifichier

L'application `FaireDesCercles`, décrite dans le fichier `FaireDesCercles.java`, utilise le type `Cercle`, défini dans le fichier `Cercle.java`. Deux fichiers distincts sont donc nécessaires à la définition d'un programme qui utilise des objets `Cercle`.

Bien que cela puisse paraître curieux pour un débutant, l'application `FaireDesCercles` s'exécute correctement, malgré cette séparation des fichiers. Examinons comment fonctionne l'ordinateur dans un tel cas.

Nous l'avons déjà observé (voir, au chapitre d'introduction, « Naissance d'un programme », la section « Exécuter un programme »), deux phases sont nécessaires pour exécuter un programme Java : la phase de compilation et la phase d'interprétation. Si l'application est conçue avec plusieurs fichiers, ces deux phases sont aussi indispensables.

Phase de compilation

Lors de la compilation d'un programme constitué de plusieurs fichiers, la question se pose de savoir comment compiler l'ensemble de ces fichiers.

Pour simplifier la tâche de la personne qui développe des applications, le compilateur Java est construit de façon à ce que seul le programme qui contient la fonction `main()` soit à compiler.

Au cours de la compilation, le compilateur constate de lui-même, au moment de la déclaration de l'objet, que l'application utilise des objets d'un type non prédéfini par le langage Java.

À partir de ce constat, il recherche, dans le répertoire où se trouve l'application qu'il compile, le fichier dont le nom correspond au nouveau type qu'il vient de détecter et dont l'extension est `java`. Tout programme Java a pour nom le nom de la classe (du type) qu'il définit.

Pour notre exemple, en compilant l'application `FaireDesCercles` grâce à la commande :

```
javac FaireDesCercles.java
```

le compilateur détecte le type `Cercle`. Il recherche alors le fichier `Cercle.java` dans le répertoire où se trouve l'application.

- Si le compilateur trouve ce fichier, il le compile aussi. En fin de compilation, deux fichiers ont été traités, `FaireDesCercles.java` et `Cercle.java`. Si le compilateur ne détecte aucune erreur, le répertoire contient les fichiers correpondant au pseudo-code et qui ont pour nom `FaireDesCercles.class` et `Cercle.class`.
- S'il ne trouve pas le fichier `Cercle.java`, il provoque une erreur de compilation du type `Class Cercle not found`.

Pour corriger cette erreur, il est possible de spécifier au compilateur où il peut trouver le fichier recherché en définissant une variable d'environnement `CLASSPATH`. Cette variable indique au compilateur quels sont les répertoires susceptibles de contenir des programmes Java. Cette définition se réalise de façon différente suivant le système utilisé, PC, Macintosh ou station Unix (voir l'annexe « Guide d'installations », section « Installation d'un environnement de développement »).

Copyright © 2016 Eyrolles.

Phase d'interprétation

Une fois le programme compilé, l'exécution du programme est réalisée grâce à l'interpréteur de la machine virtuelle Java (JVM), qui exécute le pseudo-code associé au programme contenant la fonction `main()`. Pour notre exemple, la commande est :

```
java FaireDesCercles
```

Lorsque l'interpréteur trouve en cours d'exécution, la déclaration d'un objet de type non prédéfini, il recherche, par l'intermédiaire du chargeur de classe (un programme, aussi appelé *class loader*, défini dans la JVM), le pseudo-code associé au type de l'objet et défini dans un fichier dont l'extension est `.class`. Pour notre exemple, le chargeur de classe recherche le fichier `Cercle.class`. Une fois trouvé, il charge le code en mémoire pour l'exécuter.

Analyse des résultats de l'application

Au cours des sections précédentes, nous avons observé que tout objet déclaré contenait une adresse correspondant à l'adresse où sont stockées les informations relatives à cet objet. Pour accéder aux données et méthodes de chaque objet, il suffit de passer par l'opérateur « . ».

Grâce à cette nouvelle façon de stocker l'information, les transformations d'un objet par l'intermédiaire d'une méthode de la classe sont visibles pour tous les objets de la même classe. Autrement dit, si une méthode fournit plusieurs résultats, ces modifications sont visibles en dehors de la méthode et pour toute l'application.

Pour mieux comprendre cette technique, examinons comment s'exécute le programme `FaireDesCercles`. Les valeurs grisées correspondent aux valeurs saisies par l'utilisateur.

```
Cercle centre en 0,0
de rayon : 0
Entrez la position en x : 10
Entrez la position en y : 10
Entrez le rayon : 5
```

Les valeurs saisies au clavier par l'utilisateur sont directement stockées en `A.x`, `A.y` et `A.r` grâce aux instructions `A.x = lectureClavier.nextInt()`,...

```
Cercle centre en 10, 10
de rayon : 5
```

La méthode `afficher()` est appliquée à l'objet `A` (`A.afficher()`). Elle consulte et affiche les données associées à cet objet, soit `10` pour `x` (en réalité `A.x`), `10` pour `y` (en réalité `A.y`) et `5` pour `A.r`.

```
Votre cercle a pour perimetre : 31.41592653589793
```

Copyright © 2016 Eyrolles.

De la même façon, la méthode `périmètre()` est appliquée à l'objet `A` (`A.périmètre()`). L'expression `2*Math.PI * r`, définie dans la méthode, est donc calculée pour `r` (`A.r`) valant 5.

```
Après deplacement :
Cercle centre en 5, 2
de rayon : 5
```

L'instruction `A.déplacer(5, 2)` passe les nouvelles coordonnées de la position du centre du cercle en paramètres. Les données `x` et `y` de l'objet `A` sont modifiées en conséquence (voir figure 7-7).

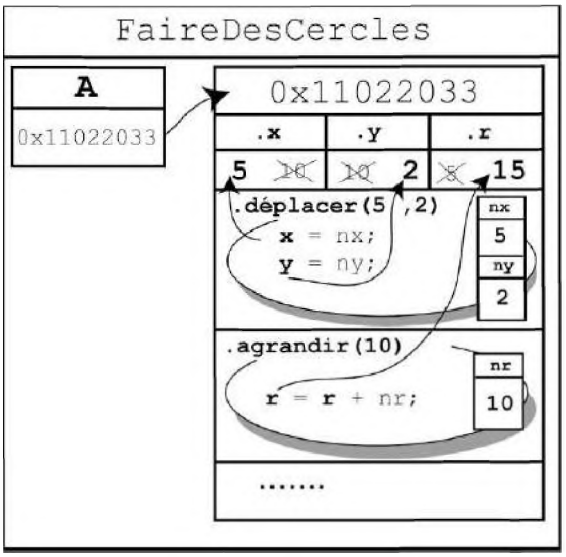


Figure 7-7 *Les méthodes appliquées à un objet exploitent les données relatives à cet objet.*

```
Apres agrandissement :
Cercle centre en 5, 2
de rayon : 15
```

L'instruction `A.agrandir(10)` passe en paramètre la valeur d'accroissement du rayon du cercle. La donnée `r` de l'objet `A` est augmentée de cette valeur (voir figure 7-7).

À chaque appel de la méthode `afficher()` appliquée à l'objet `A`, les valeurs courantes des données (`x`, `y` et `r`) de l'objet `A` sont affichées.

Observons que, lorsque l'objet `A` est déplacé, les deux coordonnées `x` et `y` sont modifiées. La méthode `déplacer()` modifie le contenu des deux variables d'instance `x` et `y` de l'objet `A`. Cette transformation est visible en dehors de l'objet lui-même, puisque la méthode `afficher()` affiche à l'écran le résultat de cette modification.

Copyright © 2016 Eyrolles.

Résumé

La classe `String` est une classe prédéfinie du langage Java, qui définit des « variables » contenant des suites de caractères (des mots ou des chaînes de caractères).

La classe `String` est un type de données composé de méthodes, qui permettent la recherche de mots ou de caractères dans un texte. Les mots peuvent aussi être comparés suivant l'ordre alphabétique ou transformés en d'autres formats.

L'étude des objets de type `String` montre qu'une classe est une association de données (information ou valeur de tout type) et de méthodes (outils d'accès et de transformation des données). Définies dans une classe, ces méthodes ne peuvent s'appliquer qu'aux données de cette même classe.

Le langage Java offre la possibilité au programmeur de développer ses propres classes. Construire une classe, c'est définir un nouveau type. Pour cela, il est nécessaire de procéder de la façon suivante :

- Déterminer les caractéristiques communes à ce que l'on souhaite décrire. Ce sont les données, les attributs, les propriétés ou encore les membres de la classe.
- Définir toutes les opérations et traitements réalisables sur ces éléments. Ces opérations sont aussi appelées méthodes, ou encore comportements.

Une classe définissant un type structuré n'est pas une application directement exécutable. Elle ne contient pas de fonction `main()`.

Les types structurés sont utilisés dans les applications en déclarant des « variables », dont le type correspond au nom de la classe définie précédemment, comme le montre l'instruction suivante :

```
TypeDeL'Objet chose = new TypeDeL'Objet();
```

L'opérateur `new` détermine l'adresse où stocker les informations relatives à la variable déclarée. Il réserve l'espace mémoire nécessaire pour stocker les données et les méthodes de la classe. Les données sont initialisées à `0` pour les entiers, à `0.0` pour les réels, à `'\0'` pour les caractères et à `null` pour tous les autres types structurés.

À cette étape, la variable est appelée un objet dans le jargon informatique. Un objet est donc un élément particulier, qui représente une classe définissant un type structuré. On dit aussi que c'est une instance de la classe. Les données (propriétés ou attributs) qui la définissent sont appelées variables d'instance.

L'accès aux variables d'instance ainsi qu'aux méthodes de la classe se fait par l'intermédiaire de l'opérateur point (`.`), comme le montre l'exemple suivant :

```
chose.nomDeLaDonnée = valeur du bon type ;
chose.nomDeLaMéthode(liste des paramètres éventuels) ;
```

en supposant que la donnée et la méthode aient été préalablement définies pour le type de l'objet `chose`.

Copyright © 2016 Eyrolles.

Exercices

Utiliser les objets de la classe String

Exercice 7.1 Écrivez un programme qui réalise les opérations suivantes :

a. Demander la saisie d'une phrase.

b. Afficher la phrase en majuscules.

c. Compter le nombre de `a` dans la phrase puis, s'il y en a, transformer tous les `a` en `*`.

d. Tester si, entre le cinquième caractère et le douzième, se trouve une séquence de caractères préalablement saisie au clavier.

Exercice 7.2 Écrivez un programme qui permet d'obtenir les actions suivantes :

a. Saisir des mots jusqu'à ce que l'utilisateur entre le mot `Fin`.

b. Afficher, parmi les mots saisis, le premier dans l'ordre alphabétique.

c. Afficher, parmi les mots saisis, le dernier dans l'ordre alphabétique.

Attention Le mot `Fin` ne doit pas être pris en compte dans la liste des mots saisis.

Exercice 7.3 Transformez le programme donné en exemple au chapitre 3 « Faire des choix », section « Calculer le jour d'un mois donné », de façon à ce que l'utilisateur entre le nom du mois dont il souhaite connaître le nombre de jours plutôt que son numéro. Pour cela, vous devrez utiliser la nouvelle structure `switch` de la version 7 de Java.

L'exécution du programme aura pour affichage :

```
De quel mois s'agit-il ? : Juin
De quelle annee ? : 1999
En 1999 le mois de juin a 30 jours
```

Ou :

```
De quel mois s'agit-il ? : février
De quelle annee ? : 2000
En 2000 le mois de février a 29 jours
```

Ou encore :

```
De quel mois s'agit-il ? : Decembre
De quelle annee ? : 2010
En 2010 le mois de décembre a 31 jours
```

Vous devez faire attention à traiter tous les cas (accent, majuscule et minuscule) pour obtenir un résultat cohérent, quel que soit le mois saisi par l'utilisateur.

Copyright © 2016 Eyrolles.

Créer une classe d'objets

Exercice 7.4 L'objectif est de définir une représentation d'un objet `Livre`.

a. Sachant qu'un livre est défini à partir de son titre, du nom et du prénom de l'auteur, d'une catégorie (Policier, Roman, Junior, Philosophie, Science-fiction), d'un numéro ISBN et d'un code d'enregistrement alphanumérique unique (voir exercice 7.4 ci-après), définissez les données de la classe `Livre`.

b. Écrivez une application `Bibliotheque` qui utilise un objet `livrePoche` de type `Livre` et qui demande la saisie au clavier du titre, des nom et prénom de l'auteur et du numéro ISBN.

Consulter les variables d'instance

Exercice 7.5 Définition des comportements d'un objet de type `Livre` :

a. Dans la classe `Livre`, décrivez la méthode `afficherUnLivre()` qui affiche les caractéristiques du livre concerné.

b. Modifiez l'application `Bibliotheque` de façon à afficher les caractéristiques de l'objet `livrePoche`.

c. Le code d'enregistrement d'un livre est construit à partir des deux premières lettres des nom et prénom de l'auteur, de la catégorie du livre et des deux derniers chiffres du code ISBN. Écrire la méthode `calculerLeCode()` qui permet de calculer ce code.

Remarque Vous pouvez utiliser la méthode `substring()` pour extraire une sous-chaîne d'un mot.

d. Modifiez l'application `Bibliotheque` de façon à calculer et afficher le code de l'objet `livrePoche`.

Analyser les résultats d'une application objet

Exercice 7.6 Pour bien comprendre ce que réalise l'application `FaireDesTriangles`, observez les deux programmes suivants :

La classe Triangle

```
import java.util.*;
public class Triangle  { // Le fichier s'appelle Triangle.java
 public int xA, yA, xB, yB, xC, yC ;

 public void créer() {
```

Copyright © 2016 Eyrolles.

```
    Scanner lectureClavier = new Scanner(System.in);
    System.out.println("Point A : ");
    System.out.print("Entrez l'abscisse  : ");
    xA = lectureClavier.nextInt();
    System.out.print("Entrez l'ordonnee : ");
    yA = lectureClavier.nextInt();
    System.out.println("Point B : ");
    System.out.print("Entrez l'abscisse  : ");
    xB = lectureClavier.nextInt();
    System.out.print ("Entrez l'ordonnee : ");
    yB = lectureClavier.nextInt();
    System.out.println("Point C : ");
    System.out.print("Entrez l'abscisse  : ");
    xC = lectureClavier.nextInt();
    System.out.print ("Entrez l'ordonnee : ");
    yC = lectureClavier.nextInt(); }

  public void afficher()  {
    System.out.println("Point A : " + xA + "  " + yA);
    System.out.println("Point B : " + xB + "  " + yB);
    System.out.println("Point C : " + xC + "  " + yC);
  }
  public void deplacer(int nx, int ny){
    xA += nx;
    yA += ny;
    xB += nx;
    yB += ny;
    xC += nx;
    yC += ny;
  }
} // Fin de la classe Triangle
```

La classe FaireDesTriangles

```
public class FaireDesTriangles {
// Le fichier s'appelle Triangle.java
public static void main(String[] args) {
Triangle T  = new Triangle();
  T.afficher();
  T.créer();
  T.afficher();
  T.deplacer(100, 0);
  T.afficher();
                                                    }
} // fin de la classe FaireDesTriangles
```

Copyright © 2016 Eyrolles.

a. Quel est le programme qui correspond à l'application ?

b. Quel est le programme définissant le type `Triangle` ?

c. Recherchez les attributs de la classe `Triangle`, et donnez leur nom.

d. Combien de méthodes sont définies dans la classe `Triangle` ? Donnez leur nom.

e. Quels sont les objets utilisés par l'application `FaireDesTriangles` ? Que valent leurs données `xA`, `yA`, `xB`, `yB`, `xC` et `yC` après exécution de l'instruction déclaration ?

f. Sur la représentation graphique ci-dessous, placez, pour l'objet `T`, les valeurs initiales ainsi que le nom des méthodes.

g. À l'appel de la méthode `créer()`, comment les valeurs sont-elles affectées aux attributs des objets concernés ? Modifiez les cases concernées sur la représentation graphique.

h. Même question pour la méthode `déplacer()`.

i. Quel est le résultat final de l'application ?

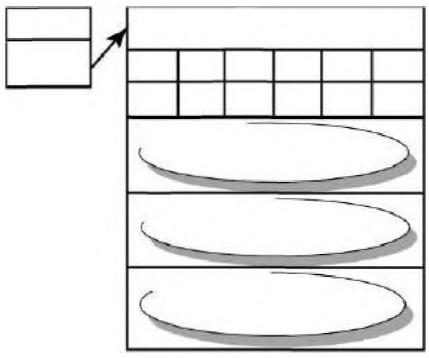


Figure 7-8 *La classe Rectangle*

Exercice 7.7 En vous inspirant de la classe `Triangle` et de l'application `FaireDesTriangles` précédentes,

a. Écrire la classe `Rectangle`, sachant que tout rectangle est une forme géométrique possédant :
une position en `X` (comprise entre `0` et `800`) et en `Y` (comprise entre `0` et `600`) ;
une hauteur dont la valeur est comprise entre `0` et `600` ;
une largeur dont la valeur est comprise entre `0` et `800` ;
Une couleur couleur variant entre `0` et `10`;

b. Définir la méthode `créer()`, `afficher()` et `déplacer()`.

c. Écrire l'application `FaireDesRectangles` qui crée un rectangle `R` en `200, 200`, de hauteur égale à `150` pixels et de largeur `100`. L'application affiche les coordonnées de `R` avant et après un déplacement de `200` sur l'axe des `Y`.

Exercice 7.8 Afin de vérifier que les valeurs saisies par la méthode `créer()`, sont valides par rapport aux contraintes de l'énoncé, reprendre la méthode `verifierAvecBornes()` de l'exercice 5.6.d et nommez la `verifier()`.

Copyright © 2016 Eyrolles.

a. Modifier la fonction de façon à ce qu'elle ne soit plus `static`.

b. Utilisez la fonction `verifier()` à l'intérieur de la méthode `créer()`, en choisissant pour chacune des propriétés les bons paramètres d'appel, de façon à ce que les valeurs saisies correspondent aux contraintes énoncées.

c. Vérifier la bonne marche de l'application en cherchant à créer un rectangle en `-10`, `-5`, de couleur égale à `20`, de hauteur et de largeur égales à `900`.

Remarque Les classes `Livre`, `Cercle`, `Triangle` et `Rectangle` traitées dans ce chapitre, font partie d'une série d'exercices pour les chapitres suivants. Pour acquérir toutes les notions abordées au cours des différents chapitres, il est conseillé de réaliser chaque exercice au fur et à mesure de l'avancée du livre

Le projet : Gestion d'un compte bancaire

Traiter les chaînes de caractères

Le `type` d'un compte et son `numéro` ne sont plus définis respectivement comme `char` et `long` mais comme deux objets de type `String`. Le type d'un compte peut donc prendre maintenant les thèmes `courant`, `joint` ou `épargne`.

a. Saisissez le `type` du compte de façon à ce que l'utilisateur entre au clavier `C`, `J` ou `E`. Le programme place dans la variable `type` les chaînes `courant`, `joint` ou `épargne` en fonction de la lettre saisie.

b. Saisissez le `numéro` de compte sous la forme d'une chaîne de caractères.

c. Transformez tous les tests faisant appel aux variables `type` et `numéro` de façon à tester non plus sur des caractères mais sur des `String`.

Définir le type Compte

Dans un fichier nommé `Compte.java`, définissez la classe `Compte` en procédant de la façon suivante :

a. Déterminez les données qui définissent tout compte bancaire.

b. Écrivez les méthodes associées, par exemple :

- `créerCpte()`, en reprenant les instructions de l'option `1`, décrites au chapitre précédent. Placez-les sous l'en-tête de la fonction qui a pour forme `public void créerCpte()`. La méthode ne possède ni paramètre, ni type de retour, car elle ne fait que modifier les données caractéristiques d'un compte déclaré en dehors.
- `afficherCpte()`, en reprenant la fonction écrite au chapitre précédent et en supprimant le mot-clé `static`. Les variables `num`, `type`, `taux` et `val` ne sont à déclarer ni

Copyright © 2016 Eyrolles.

à l'intérieur, ni en paramètre de la méthode. Elles sont définies comme données de la classe `compte`, en dehors de la méthode.

Construire l'application Projet

Dans un fichier nommé `Projet.java`, écrivez l'application contenant la fonction `main()` en procédant de la façon suivante :

a. Faites appel aux fonctions `alAide()`, `sortir()` et `menuPrincipal()`.

b. Créez un objet de type `Compte` grâce à l'instruction de déclaration `Compte c = new Compte();`.

c. Dans les options appropriées du menu, appelez les méthodes de la classe `Compte`, comme `c.créerCpte()` ou `c.afficherCpte()`.

d. À l'exécution du programme, observez que la méthode `afficherCpte()` affiche les différentes valeurs du compte, modifiées par la méthode `créerCpte()`. Une méthode par l'intermédiaire d'un objet peut, par conséquent, transmettre plusieurs résultats.

Définir le type LigneComptable

Dans un fichier nommé `LigneComptable.java`, définissez la classe `LigneComptable` en procédant de la façon suivante :

a. Déterminez les données qui définissent toute ligne comptable.

Pour en savoir plus Voir, au chapitre introductif, « Naissance d'un programme », la définition de l'option *3*, à la section « Le projet : Gestion d'un compte bancaire ? ».

b. Écrivez les méthodes associées, par exemple :

- `créerLigneComptable()`, qui demande la saisie au clavier des valeurs correspondant aux données de la classe `LigneComptable`.
- `afficherLigne()`, qui affiche les données caractéristiques d'une ligne comptable.

Modifier le type Compte

Dans le fichier `Compte.java` :

a. Définissez une nouvelle donnée (variable d'instance) décrivant une ligne comptable, en écrivant la déclaration `LigneComptable ligne ;` au même niveau que `type`, `numéro`, etc.

b. Écrivez la méthode `créerLigne()` qui permette les actions suivantes :

Copyright © 2016 Eyrolles.

- créer en mémoire l'objet `ligne` grâce à l'instruction de déclaration `ligne = new LigneComptable() ;`
- faire appel à la méthode `créerLigneComptable()` par l'intermédiaire de l'objet ligne de façon à enregistrer les valeurs numériques associées à la ligne créée ;
- modifier la valeur courante du compte à partir de la valeur (débit ou crédit) saisie dans la méthode `créerLigneComptable()`.

c. Modifiez la méthode `afficherCpte()` de façon à afficher les informations stockées dans `ligne`, en utilisant l'instruction `ligne.afficherLigne()`.

Modifier l'application Projet

Dans le fichier nommé `Projet.java`, modifiez l'option `3` de l'application de façon qu'une ligne comptable soit créée pour le compte `C`, défini à l'option `1`.

a. À l'exécution de l'application, que se passe-t-il si l'utilisateur ayant créé un compte affiche ce dernier sans avoir jamais créé de ligne comptable ? Pourquoi ?

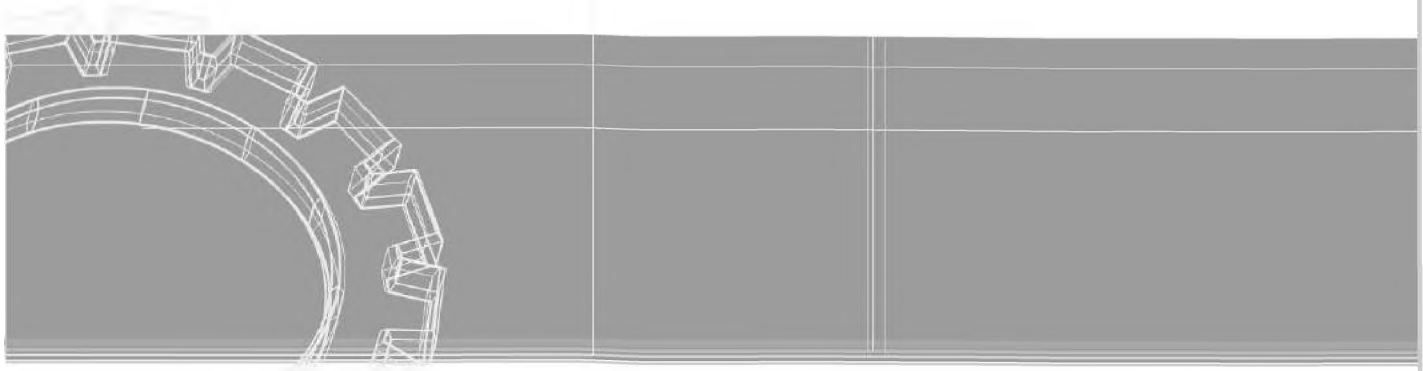
b. Comment faire pour remédier à cette situation ?

Copyright © 2016 Eyrolles.

Copyright © 2016 Eyrolles.

Chapitre 8

Les principes du concept objet



Au cours du chapitre précédent, nous avons examiné comment mettre en place des objets à l'intérieur d'un programme Java. Cette étude a montré combien la structure générale des programmes se trouvait modifiée par l'emploi des objets.

En réalité, les objets sont beaucoup plus qu'une structure syntaxique. Ils sont régis par des principes essentiels, qui constituent les fondements de la programmation objet. Dans ce chapitre, nous étudions avec précision l'ensemble de ces principes.

Nous déterminons d'abord (section « La communication objet ») les caractéristiques d'une donnée `static` et évaluons leurs conséquences sur la construction des objets en mémoire. Nous analysons également la technique du passage de paramètres par référence. Nous observons qu'il est possible, avec la technologie objet, qu'une méthode transmette plusieurs résultats à une autre méthode.

Nous expliquons ensuite (section « Les objets contrôlent leur fonctionnement »), le concept d'encapsulation des données, et nous examinons pourquoi et comment les objets protègent leurs données.

Enfin, nous définissons (section « L'héritage ») la notion d'héritage entre classes. Nous observons combien cette notion est utile puisqu'elle permet de réutiliser des programmes tout en apportant des variations dans le comportement des objets héritants.

Copyright © 2016 Eyrolles.

La communication objet

En définissant un type ou une classe, le développeur crée un modèle, qui décrit les fonctionnalités des objets utilisés par le programme. Les objets sont créés en mémoire à partir de ce modèle, par copie des données et des méthodes.

Cette copie est réalisée lors de la réservation des emplacements mémoire grâce à l'opérateur `new`, qui initialise les données de l'objet et fournit, en retour, l'adresse où se trouvent les informations stockées.

La question est de comprendre pourquoi l'interpréteur réalise cette copie en mémoire, alors que cela lui était impossible auparavant.

Les données static

La réponse à cette interrogation se trouve dans l'observation des différents programmes proposés dans ce manuel (voir les chapitres 6, « Fonctions, notions avancées », et 7, « Les classes et les objets »). Comme nous l'avons déjà constaté (voir, au chapitre précédent, la section « Construire et utiliser ses propres classes »), le mot-clé `static` n'est plus utilisé lors de la description d'un type, alors qu'il était présent dans tous les programmes précédant ce chapitre.

C'est donc la présence ou l'absence de ce mot-clé qui fait que l'interpréteur construise ou non des objets en mémoire.

Remarque

Lorsque l'interpréteur rencontre le mot-clé `static` devant une variable ou une méthode, il réserve un seul et unique emplacement mémoire pour y stocker la valeur ou le pseudo-code associé. Cet espace mémoire est communément accessible pour tous les objets du même type.

Lorsque le mot-clé `static` n'apparaît pas, l'interpréteur réserve, à chaque appel de l'opérateur `new`, un espace mémoire pour y charger les données et les pseudo-codes décrits dans la classe.

Exemple : compter des cercles

Pour bien comprendre la différence entre une donnée `static` et une donnée non `static`, nous allons modifier la classe `Cercle`, de façon qu'il soit possible de connaître le nombre d'objets `Cercle` créés en cours d'application.

Pour ce faire, l'idée est d'écrire une méthode `créer()` qui permet d'une part, de saisir des valeurs `x`, `y` et `r` pour chaque cercle à créer et d'autre part, d'incrémenter un compteur de cercles.

La variable représentant ce compteur doit être indépendante des objets créés, de sorte que sa valeur ne soit pas réinitialisée à zéro à chaque création d'objet. Cette variable doit cependant être accessible pour chaque objet de façon qu'elle puisse s'incrémenter de `1` à chaque appel de la méthode `créer()`.

Copyright © 2016 Eyrolles.

Pour réaliser ces contraintes, le compteur de cercles doit être une variable de classe, c'est-à-dire une variable déclarée avec le mot-clé `static`. Examinons tout cela dans le programme suivant.

```
import java.util.*;
public class Cercle  {
 public int x, y, r ; // position du centre et rayon
 public static int nombre; // nombre de cercle

  public void créer()  {
   Scanner lectureClavier = new Scanner(System.in);
   System.out.print(" Position en x :  ");
   x = lectureClavier.nextInt();
   System.out.print(" Position en y :  ");
   y = lectureClavier.nextInt();
   System.out.print(" Rayon            :  ");
   r = lectureClavier.nextInt();
   nombre ++;
  }
  // et toutes les autres méthodes de la classe Cercle définies au
  // chapitre précédent
} // Fin de la classe Cercle
```

Les données définies dans la classe `Cercle` sont de deux sortes : les variables d'instance `x`, `y` et `r`, et la variable de classe `nombre`. Seul le mot-clé `static` permet de différencier leur catégorie.

Grâce au mot-clé `static`, la variable de classe `nombre` est un espace mémoire commun, accessible pour tous les objets créés. Pour faire appel à cette variable, il suffit de l'appeler par son nom véritable c'est-à-dire `nombre`, si elle est utilisée dans la classe `Cercle`, ou `Cercle.nombre`, si elle est utilisée en dehors de cette classe.

Pour en savoir plus Voir, au chapitre 6, « Fonctions, notions avancées », la section « Variable de classe ».

Exécution de l'application CompterDesCercles

Pour mieux saisir la différence entre les variables d'instance (non `static`) et les variables de classe (`static`), observons comment fonctionne l'application `CompterDesCercles`.

```
public class CompterDesCercles  {
 public static void main(String [] arg)
 {
  Cercle A  = new Cercle();
  A.créer();
  System.out.println("Nombre de cercles : " + Cercle.nombre);

  Cercle B = new Cercle();
  B.créer();
  System.out.println("Nombre de cercles : " + Cercle.nombre);
 }
} // Fin de la classe CompterDesCercles
```

Copyright © 2016 Eyrolles.

Dans ce programme, deux objets de type `Cercle` sont créés à partir du modèle défini par le type `Cercle`. Chaque objet est un représentant particulier, ou une instance, de la classe `Cercle`, de position et de rayon spécifiques.

Lorsque l'objet `A` est créé en mémoire grâce à l'opérateur `new`, les données `x`, `y` et `r` sont initialisées à `0` au moment de la réservation de l'espace mémoire. La variable de classe `nombre` est elle aussi créée en mémoire, et sa valeur est également initialisée à `0`.

Lors de l'exécution de l'instruction `A.créer();`, les valeurs des variables `x`, `y` et `r` de l'instance `A` sont saisies au clavier (`x = lectureClavier.nextInt()`, ...). La variable de classe `nombre` est incrémentée de `1` (`nombre++`). Le nombre de cercles est alors de `1` (voir l'objet `A`, décrit à la figure 8-1).

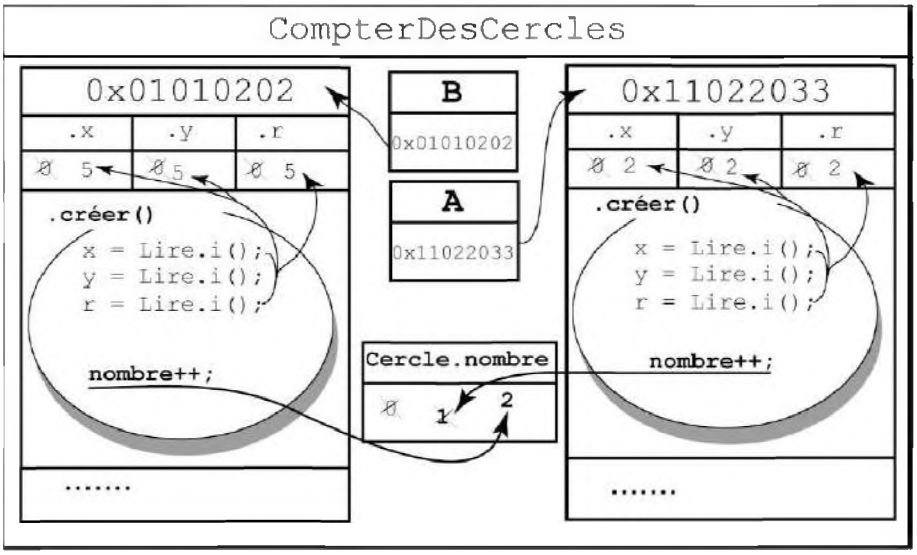


Figure 8-1 *La variable de classe Cercle.nombre est créée en mémoire, avec l'objet A. Grâce au mot-clé static, il y a, non pas réservation d'un nouvel espace mémoire (pour la variable nombre) lors de la création de l'objet B, mais préservation de l'espace mémoire ainsi que de la valeur stockée.*

De la même façon, l'objet `B` est créé en mémoire grâce à l'opérateur `new`. Les données `x`, `y` et `r` sont, elles aussi, initialisées à `0`.

Pour la variable de classe `nombre` en revanche, cette initialisation n'est pas réalisée. La présence du mot-clé `static` fait que la variable de classe `nombre`, qui existe déjà en mémoire, ne peut être réinitialisée directement par l'interpréteur.

Il y a donc, non pas réservation d'un nouvel emplacement mémoire, mais préservation du même emplacement mémoire avec conservation de la valeur calculée à l'étape précédente, soit `1`.

Copyright © 2016 Eyrolles.

Après saisie des données `x`, `y` et `r` de l'instance `B`, l'instruction `nombre++` fait passer la valeur de `Cercle.nombre` à `2` (voir l'objet `B` décrit à la figure 8-1).

N'existant qu'en un seul exemplaire, la variable de classe `nombre` permet le comptage du nombre de cercles créés. L'incrémentation de cette valeur est réalisée indépendamment de l'objet, la variable étant commune à tous les objets créés.

Le passage de paramètres par référence

La communication des données entre les objets passe avant tout par l'intermédiaire des variables d'instance. Nous l'avons observé à la section précédente, lorsqu'une méthode appliquée à un objet modifie les valeurs de plusieurs données de cet objet, cette modification est visible en dehors de la méthode et de l'objet lui-même.

Il existe cependant une autre technique qui permet la modification des données d'un objet : le passage de **paramètres par référence**.

Ce procédé est utilisé lorsqu'on passe en paramètre d'une méthode, non plus une simple variable (de type `int`, `char` ou `double`), mais un objet. Dans cette situation, l'objet étant défini par son adresse (référence), la valeur passée en paramètre n'est plus la valeur réelle de la variable mais l'adresse de l'objet.

Grâce à cela, les modifications apportées sur l'objet passé en paramètre et réalisées à l'intérieur de la méthode sont visibles en dehors même de la méthode.

Échanger la position de deux cercles

Pour comprendre en pratique le mécanisme du passage de paramètres par référence, nous allons écrire une application qui échange la position des centres de deux cercles donnés.

Pour cela, nous utilisons le mécanisme d'échange de valeurs, en l'appliquant à la coordonnée `x` puis à la coordonnée `y` des centres des deux cercles à échanger.

Pour en savoir plus Les mécanismes d'échange de valeurs sont définis au chapitre 1, « Stocker une information ».

Examinons la méthode `échanger()`, dont le code ci-dessous s'insère dans la classe `Cercle`.

Pour en savoir plus La classe `Cercle` est définie au chapitre 7, « Les classes et les objets », la section « La classe descriptive du type Cercle ».

```
public void échanger(Cercle autre)  {    // Échange la position d'un
  int tmp;          // cercle avec celle du cercle donné en paramètre
  tmp = x;          // Échanger la position en x
  x = autre.x;
```

Copyright © 2016 Eyrolles.

```
    autre.x = tmp;
    tmp = y;          // Échanger la position en y
    y = autre.y;
    autre.y = tmp;
}
```

Pour échanger les coordonnées des centres de deux cercles, la méthode `échanger()` doit avoir accès aux valeurs des coordonnées des deux centres des cercles concernés.

Si par exemple, la méthode est appliquée au cercle `B` (`B.échanger()`), ce sont les variables d'instance `x` et `y` de l'objet `B` qui sont modifiées par les coordonnées du centre du cercle `A`. La méthode doit donc connaître les coordonnées du cercle `A`. Pour ce faire, il est nécessaire de passer ces valeurs en paramètres de la fonction.

La technique consiste à passer en paramètres, non pas les valeurs `x` et `y` du cercle avec lequel l'échange est réalisé, mais un objet de type `Cercle`. Dans notre exemple, ce paramètre s'appelle `autre`. C'est le paramètre formel de la méthode représentant n'importe quel cercle, et il peut donc représenter par exemple, le cercle `A`.

Le fait d'échanger les coordonnées des centres de deux cercles revient à échanger les coordonnées du couple (`x`, `y`) du cercle sur lequel on applique la méthode (`B.x`, `B.y`) avec les coordonnées (`autre.x`, `autre.y`) du cercle passé en paramètre de la méthode (`A.x`, `A.y`).

Examinons maintenant comment s'opère effectivement l'échange en exécutant l'application suivante :

```
public class EchangerDesCercles  {
 public static void main(String [] arg)  {
  Cercle A  = new Cercle();
  A.créer();
  System.out.println("Le cercle A : ");
  A.afficher();

  Cercle B = new Cercle();
  B.créer();
  System.out.println("Le cercle B : ");
  B.afficher() ;

  B.échanger(A) ;
  System.out.println("Apres echange, ") ;
  System.out.println("Le cercle A : ") ;
  A.afficher() ;
  System.out.println("Le cercle B : ") ;
  B.afficher() ;
 }
}
```

Copyright © 2016 Eyrolles.

Exécution de l'application EchangerDesCercles

Nous supposons que l'utilisateur ait saisi les valeurs suivantes, pour le cercle `A` :

```
Position en x : 2
Position en y : 2
Rayon          : 2
Le cercle A :
Centre en 2, 2
Rayon : 2
et pour le cercle B :
Position en x : 5
Position en y : 5
Rayon          : 5
Le cercle B :
Centre en 5, 5
Rayon : 5
```

L'instruction `B.échanger(A)` échange les coordonnées (`x`, `y`) de l'objet `B` avec celles de l'objet `A`. C'est donc le pseudo-code de l'objet `B` qui est interprété, comme illustré à la figure 8-2.

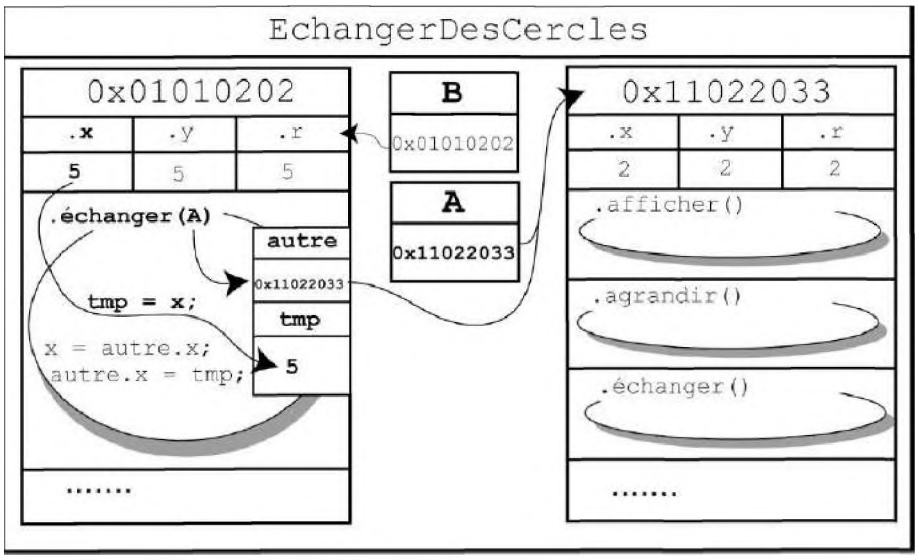


Figure 8-2 *L'instruction B.échanger(A) fait appel à la méthode échanger() de l'objet B. Les données x, y et r utilisées par cette méthode sont celles de l'objet B.*

Examinons le tableau d'évolution des variables déclarées pour le pseudo-code de l'objet `B`.

Instruction	tmp	x	y	autre
Valeurs initiales	–	5	5	0x11022033

Copyright © 2016 Eyrolles.

- À l'entrée de la méthode, la variable `tmp` est déclarée sans être initialisée.
- La méthode est appliquée à l'objet `B`. Les variables `x` et `y` de l'instance `B` ont pour valeurs respectives `5` et `5`.
- L'objet `autre` est simplement déclaré en paramètre de la fonction `échanger(Cercle autre)`. L'opérateur `new` n'étant pas appliqué à cet objet, aucun espace mémoire supplémentaire n'est alloué.

  Comme `autre` représente un objet de type `Cercle`, il ne peut contenir qu'une adresse et non pas une simple valeur numérique. Cette adresse est celle du paramètre effectivement passé lors de l'appel de la méthode.

  Pour notre exemple, l'objet `A` est passé en paramètre de la méthode `(B.échanger(A))`. La case mémoire de la variable `autre` prend donc pour valeur l'adresse de l'objet `A`.

instruction	Tmp	x	autre	autre.x (A.x)
`tmp = x ;`	5	5	0x11022033	2
`x = autre.x ;`	5	2	0x11022033	2
`autre.x = tmp ;`	5	2	0x11022033	5

- La variable `tmp` prend ensuite la valeur de la coordonnée `x` de l'objet `B`, soit `5`.

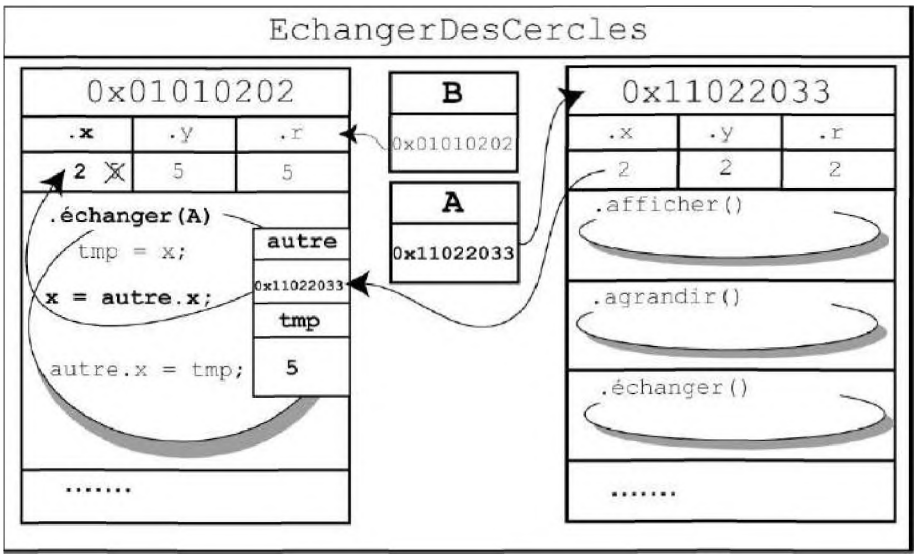


Figure 8-3 *L'objet autre est le paramètre formel de la méthode échanger(). En écrivant B.échanger(A), l'objet autre stocke la référence mémorisée en A. De cette façon, autre.x représente également A.x. La variable x de l'instance B prend la valeur de A.x grâce à l'instruction x = autre.x.*

Copyright © 2016 Eyrolles.

- Lorsque l'instruction `x = autre.x` est exécutée, la coordonnée `x` de l'objet `B` prend la valeur de la coordonnée `x` de l'objet `autre.x`. Puisque `autre` correspond à l'adresse de l'objet `A`, le fait de consulter le contenu de `autre.x` revient en réalité, à consulter le contenu de `A.x` (voir figure 8-3). La variable d'instance `A.x` contenant la valeur 2, `x` (`B.x`) prend la valeur 2.
- Pour finir l'échange sur les abscisses, la donnée `autre.x` prend la valeur stockée dans `tmp`. Comme `autre` et `A` correspondent à la même adresse, modifier `autre.x`, c'est aussi modifier `A.x` (voir figure 8-4). Une fois exécuté `autre.x = tmp`, la variable `x` de l'instance `A` vaut par conséquent 5.

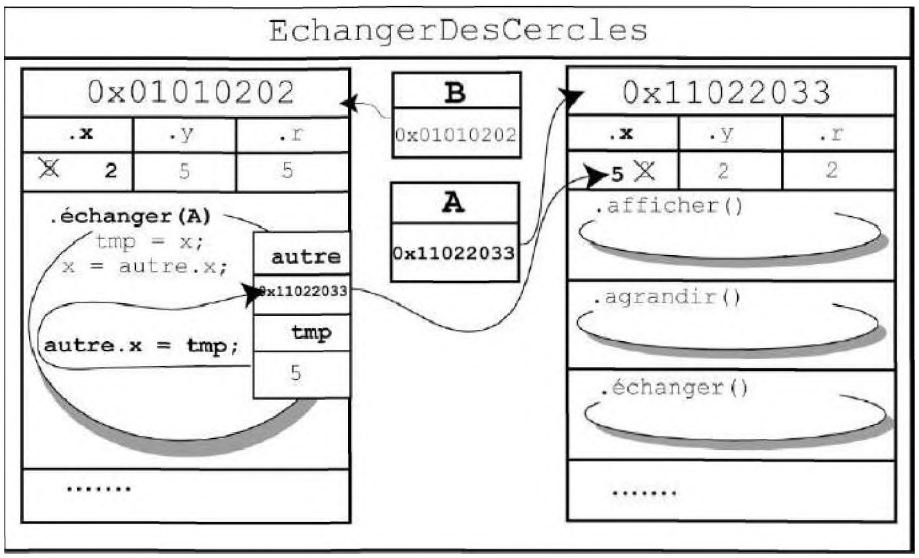


Figure 8-4 *autre et A définissent la même référence, ou adresse. C'est pourquoi le fait de modifier autre.x revient aussi à modifier A.x. Ainsi, l'instruction autre.x = tmp fait que A.x prend la valeur stockée dans tmp.*

L'ensemble de ces opérations est ensuite réalisé sur la coordonnée `y` des cercles `B` et `A` via `autre`.

instruction	tmp	y	autre	autre.y (A.y)
`tmp = y ;`	5	5	0x11022033	2
`y = autre.y ;`	5	2	0x11022033	2
`autre.y = tmp ;`	5	2	0x11022033	5

Copyright © 2016 Eyrolles.

L'exécution finale du programme a pour résultat :

```
Apres echange,
Le cercle A :
Centre en 5, 5
Rayon : 2
Le cercle B :
Centre en 2, 2
Rayon : 5
```

Au final nous constatons, à l'observation des tableaux d'évolution des variables, que les données `x` et `y` de `B` ont pris la valeur des données `x` et `y` de `A`, soit 2 pour `x` et 2 pour `y`. Parallèlement, le cercle `A` a été transformé par l'intermédiaire de la référence stockée dans `autre` et a pris les coordonnées `x` et `y` du cercle `B`, soit 5 pour `x` et 5 pour `y`.

Remarque Grâce à la technique du passage de paramètres par référence, tout objet passé en paramètre d'une méthode voit, en sortie de la méthode, ses données transformées par la méthode. Cette transformation est alors visible pour tous les objets de l'application.

Les objets contrôlent leur fonctionnement

L'un des objectifs de la programmation objet est de simuler, à l'aide d'un programme informatique, la manipulation des objets réels par l'être humain. Les objets réels forment un tout, et leur manipulation nécessite la plupart du temps un outil, ou une interface, de communication.

Par exemple, quand nous prenons un ascenseur, nous appuyons sur le bouton d'appel pour ouvrir les portes ou pour nous rendre jusqu'à l'étage désiré. L'interface de communication est ici le bouton d'appel. Nul n'aurait l'idée de prendre la télécommande de sa télévision pour appeler un ascenseur.

De la même façon, la préparation d'une omelette nécessite de casser des œufs. Pour briser la coquille d'un œuf, nous pouvons utiliser l'outil couteau. Un marteau pourrait être également utilisé, mais son usage n'est pas vraiment adapté à la situation.

Comme nous le constatons à travers ces exemples, les objets réels sont manipulés par l'intermédiaire d'interfaces **appropriées**. L'utilisation d'un outil inadapté fait que l'objet ne répond pas à nos attentes ou qu'il se brise définitivement.

Tout comme nous manipulons les objets réels, les applications informatiques manipulent des objets virtuels, définis par le programmeur. Cette manipulation nécessite des outils aussi bien adaptés que nos outils réels. Sans contrôle sur le bien-fondé d'une manipulation, l'application risque de fournir de mauvais résultats ou pire, de cesser brutalement son exécution.

Copyright © 2016 Eyrolles.

La notion d'encapsulation

Pour réaliser l'adéquation entre un outil et la manipulation d'un objet, la programmation objet utilise le concept d'**encapsulation**.

Remarque

Par ce terme, il faut entendre que les données d'un objet sont protégées, tout comme le médicament est protégé par la fine pellicule de sa capsule. Grâce à cette protection, il ne peut y avoir de transformation involontaire des données de l'objet.

L'encapsulation passe par le contrôle des données et des comportements de l'objet. Ce contrôle est établi à travers la protection des données (voir la section suivante), l'accès contrôlé aux données (voir la section « Les méthodes d'accès aux données ») et la notion de constructeur de classe (voir la section « Les constructeurs »).

La protection des données

Le langage Java fournit les niveaux de protection suivants pour les membres d'une classe (données et méthodes) :

- **Protection** `public`. Les membres (données et méthodes) d'une classe déclarés `public` sont accessibles pour tous les objets de l'application. Les données peuvent être modifiées par une méthode de la classe, d'une autre classe ou depuis la fonction `main()`.
- **Protection** `private`. Les membres de la classe déclarés `private` ne sont accessibles que pour les méthodes de la même classe. Les données ne peuvent être initialisées ou modifiées que par l'intermédiaire d'une méthode de la classe. Les données ou méthodes ne peuvent être appelées par la fonction `main()`.
- **Protection** `protected`. Tout comme les membres privés, les membres déclarés `protected` ne sont accessibles que pour les méthodes de la même classe. Ils sont aussi accessibles par les fonctions membres d'une sous-classe (voir la section « L'héritage »).

Par défaut, lorsque les données sont déclarées sans type de protection, leur protection est `public`. Les données sont alors accessibles depuis toute l'application.

Protéger les données d'un cercle

Pour protéger les données de la classe `Cercle`, il suffit de remplacer le mot-clé `public` précédant la déclaration des variables d'instance par le mot `private`. Observons la nouvelle classe, `CerclePrive`, dont les données sont ainsi protégées.

```
public class CerclePrive
{
 private int x, y, r ; // position du centre et rayon

 public void afficher() {
  // voir la section "La classe descriptive du type Cercle" du chapitre
  //"Les classes et les objets"
 }
```

Copyright © 2016 Eyrolles.

```
 public double périmètre() {
  // voir la section "La classe descriptive du type Cercle"  du chapitre
  //"Les classes et les objets"
 }
 public void déplacer(int nx, int ny) {
  // voir la section "La classe descriptive du type Cercle"  du chapitre
  //"Les classes et les objets"
 }

 public void agrandir(int nr) {
  // voir la section "La classe descriptive du type Cercle" du chapitre
  //"Les classes et les objets"
 }
} // Fin de la classe CerclePrive
```

Les données `x`, `y` et `r` de la classe `CerclePrive` sont protégées grâce au mot-clé `private`. Étudions les conséquences d'une telle protection sur la phase de compilation de l'application `FaireDesCerclesPrives`.

```
import java.util.*;
public class FaireDesCerclesPrives
{
 public static void main(String [] arg)
 {
  Scanner lectureClavier = new Scanner(System.in);
  CerclePrive A  = new CerclePrive();
  A.afficher();
  System.out.println(" Entrez le rayon : ");
  A.r = lectureClavier.nextInt();
  System.out.println(" Le cercle est de rayon : " + A.r) ;
 }
}
```

Compilation de l'application FaireDesCerclesPrives

Les données `x`, `y` et `r` de la classe `CerclePrive` sont déclarées privées. Par définition, ces données ne sont donc pas accessibles en dehors de la classe où elles sont définies.

Or, en écrivant dans la fonction `main()` l'instruction `A.r = lectureClavier.next-Int();`, le programmeur demande d'accéder depuis la classe `FaireDesCerclesPrives` à la valeur de `r`, de façon à la modifier. Cet accès est impossible, car `r` est défini en mode `private` dans la classe `CerclePrive` et non dans la classe `FaireDesCerclesPrives`.

Copyright © 2016 Eyrolles.

C'est pourquoi le compilateur détecte l'erreur `Variable r in class CerclePrive not accessible from class FaireDesCerclesPrives`.

Question Que se passe-t-il si l'on place le terme `private` devant la méthode `afficher()` ?

Réponse Lors de la compilation du fichier `FaireDesCerclesPrivés`, le message d'erreur `afficher() has private access in CerclePrive` s'affiche.

En effet, si la méthode `afficher()` est définie en `private`, elle n'est plus accessible depuis l'extérieur de la classe `CerclePrivé`. Il n'est donc pas possible de l'appeler depuis la fonction `main()` définie dans la classe `FaireDesCerclesPrivés`.

Les méthodes d'accès aux données

Lorsque les données sont totalement protégées, c'est-à-dire déclarées `private` à l'intérieur d'une classe, elles ne sont plus accessibles depuis une autre classe ou depuis la fonction `main()`. Pour connaître ou modifier la valeur d'une donnée, il est nécessaire de créer, à l'intérieur de la classe, des méthodes d'accès à ces données.

Les données privées ne peuvent être consultées ou modifiées que par des méthodes de la classe où elles sont déclarées.

De cette façon, grâce à l'accès aux données par l'intermédiaire de méthodes appropriées, l'objet permet, non seulement la consultation de la valeur de ses données, mais aussi l'autorisation ou non, suivant ses propres critères, de leur modification.

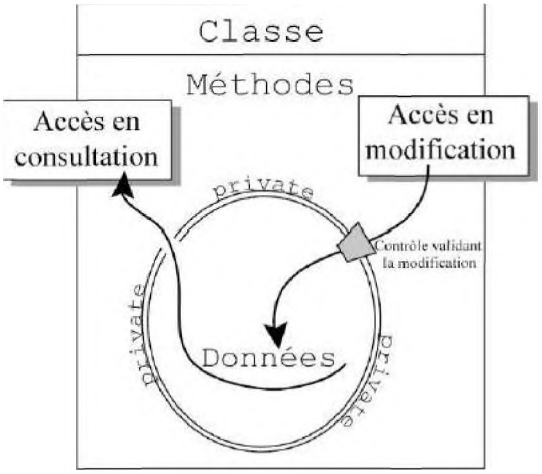


Figure 8-5 *Lorsque les données d'un objet sont protégées, l'objet possède ses propres méthodes, qui permettent soit de consulter la valeur réelle de ses données, soit de modifier les données. La validité de ces modifications est contrôlée par les méthodes définies dans la classe.*

Copyright © 2016 Eyrolles.

Remarque

Les méthodes d'une classe réalisent les modes d'accès suivants :

- **Accès en consultation.** La méthode fournit la valeur de la donnée mais ne peut la modifier. Ce type de méthode est aussi appelé **accesseur** en consultation.

- **Accès en modification.** La méthode modifie la valeur de la donnée. Cette modification est réalisée après validation par la méthode. On parle aussi d'accesseur en modification.

Contrôler les données d'un cercle

Dans l'exemple suivant, nous prenons pour hypothèse que le rayon d'un cercle ne peut jamais être négatif ni dépasser la taille de l'écran. Ces conditions doivent être vérifiées pour toutes les méthodes qui peuvent modifier la valeur du rayon d'un cercle.

Comme nous l'avons déjà observé (voir, au chapitre 7, « Les classes et les objets », la section « Quelques observations »), les méthodes `afficher()` et `périmètre()` ne font que consulter le contenu des données `x`, `y` et `r`.

Les méthodes `déplacer()`, `agrandir()` et `créer()`, en revanche, modifient le contenu des données `x`, `y` et `r`. La méthode `déplacer()` n'ayant pas d'influence sur la donnée `r`, seules les méthodes `agrandir()` et `créer()` doivent contrôler la valeur du rayon, de sorte que cette dernière ne puisse être négative ou supérieure à la taille de l'écran. Examinons la classe `CercleControle` suivante, qui prend en compte ces nouvelles contraintes :

```
import java.util.*;
public class CercleControle  {
  private int x, y, r ; // position du centre et rayon
  public void créer()    {
   Scanner lectureClavier = new Scanner(System.in);
   System.out.print(" Position en x :  ");
    x = lectureClavier.nextInt();
   System.out.print(" Position en y :  ");
    y = lectureClavier.nextInt();
   do  {
     System.out.print(" Rayon            :  ");
  r = lectureClavier.nextInt();   } while ( r < 0 || r >  600);
  }

 public void afficher()   {  //Affichage des données de la classe
  System.out.println(" Centre en " + x + ", " + y);
  System.out.println(" Rayon : " + r);
 }

 public void agrandir(int nr)   {
   if (r + nr < 0) r = 0;
```

Copyright © 2016 Eyrolles.

```
    else if ( r + nr > 600) r = 600;
    else r = r + nr;
  }
} // Fin de la classe CercleControle
```

La méthode `créer()` contrôle la valeur du rayon lors de sa saisie, en demandant de saisir une valeur pour le rayon tant que la valeur saisie est négative ou plus grande que `600` (taille supposée de l'écran). Dès que la valeur saisie est comprise entre `0` et `600`, la fonction `créer()` cesse son exécution. À la sortie de cette fonction, nous sommes certains que le rayon est compris entre `0` et `600`.

De la même façon, la méthode `agrandir()` autorise que la valeur du rayon soit augmentée de la valeur passée en paramètre, à condition que cette augmentation ne dépasse pas la taille de l'écran ou que la diminution n'entraîne pas un rayon négatif, si la valeur passée en paramètre est négative. Dans ces deux cas, la valeur du rayon est forcée respectivement à la taille de l'écran ou à `0`.

Exécution de l'application FaireDesCerclesControles

Pour vérifier que tous les objets `Cercle` contrôlent bien la valeur de leur rayon, examinons l'exécution de l'application suivante :

```
import java.util.*;
public class FaireDesCerclesControles {
 public static void main(String [] arg) {
  Scanner lectureClavier = new Scanner(System.in);
  CercleControle A  = new CercleControle();
  A.créer();
  A.afficher();
  System.out.print("Entrer une valeur d'agrandissement :");
  int plus = lectureClavier.nextInt();
  A.agrandir(plus);
  System.out.println("Apres agrandissement : ");
  A.afficher();
 }
}
```

L'objet `A` est créé en mémoire grâce à l'opérateur `new`. La valeur du rayon est initialisée à `0`. À l'appel de la méthode `créer()`, les variables d'instance `x` et `y` sont saisies au clavier, comme suit :

```
Position en x : 5
Position en y : 5
```

Copyright © 2016 Eyrolles.

Ensuite, si l'utilisateur saisit pour le rayon une valeur négative :

```
Rayon          : -3
```

ou supérieure à 600

```
Rayon          : 654
```

le programme demande de nouveau de saisir une valeur pour le rayon. L'application cesse cette répétition lorsque l'utilisateur entre une valeur comprise entre 0 et 600, comme suit :

```
Rayon          : 200
Centre en 5, 5
Rayon : 200
```

Après affichage des données du cercle A, le programme demande :

```
Entrer une valeur d'agrandissement : 450
```

La valeur du rayon vaut 200 + 450, soit 650. Ce nouveau rayon étant supérieur à 600, la valeur du rayon est bloquée par le programme à 600. L'affichage des données fournit

```
Apres agrandissement :
Centre 5, 5
Rayon : 600
```

Convention de nommage

En programmation objet, les conventions stipulent que le nom des méthodes d'accès doit être donné en suivant une règle particulière :

- Les méthodes d'accès en lecture (consultation) doivent commencer par `get`.
- Les méthodes d'accès en écriture (modification) doivent commencer par `set`.
- Derrière les termes `set` ou `get`, suit obligatoirement le nom de la propriété dont la première lettre est en majuscule.

Par exemple la méthode qui fournit la valeur du rayon s'écrit, dans la classe `Cercle` :

```
// Méthode d'accès en lecture
public int getRayon(){
    return rayon;
  }
```

La méthode qui autorise la modification du périmètre s'écrit :

```
// Méthode d'accès en écriture
public void setRayon(int r){
    rayon = r;
  }
```

Copyright © 2016 Eyrolles.

L'accès aux données d'un cercle dans l'application `FaireDesCerclesControles` s'écrit alors :

```
// Changer la valeur du rayon
A.setRayon(10);
// Afficher (consulter) la nouvelle valeur du rayon
System.out.println("Apres modification : " + A.getRayon());
```

Utiliser cette convention de nommage simplifie la lecture du code. En effet, en un seul coup d'œil, nous sommes en mesure de savoir quelle propriété de l'objet est consultée (`get`) ou modifiée (`set`), par simple lecture du nom de la propriété, dans le nom de la fonction.

La notion de constante

D'une manière générale en programmation objet, les variables d'instance ne sont que très rarement déclarées en `public`. Pour des raisons de sécurité, tout objet se doit de contrôler les transformations opérées par l'application sur lui-même. C'est pourquoi les données d'une classe sont le plus souvent déclarées en mode `private`.

Il existe des données, appelées **constantes** qui, parce qu'elles sont importantes, doivent être visibles par toutes les méthodes de l'application. Ces données sont déclarées en mode `public`. Du fait de leur invariabilité, l'application ne peut modifier leur contenu.

Pour notre exemple, la valeur `600`, correspondant à la taille (largeur et hauteur) supposée de l'écran, peut être considérée comme une donnée constante de l'application.

Il suffit de déclarer les variables « constantes » à l'aide du mot-clé `final`. Ainsi, la taille de l'écran peut être définie de la façon suivante :

```
public final int TailleEcran = 600 ;
```

Notons que la taille de l'écran est une valeur indépendante de l'objet `Cercle`. Quelle que soit la forme à dessiner (carré, cercle, etc.), la taille de l'écran est toujours la même. C'est pourquoi il est logique de déclarer la variable `TailleEcran` comme constante de classe à l'aide du mot-clé `static`.

```
public final static int TailleEcran = 600 ;
```

De cette façon, la variable `TailleEcran` est accessible en consultation depuis toute l'application, mais elle ne peut en aucun cas être modifiée, étant déclarée `final`.

Les méthodes `créer()` et `agrandir()` s'écrivent alors de la façon suivante :

```
public void créer()  {
    Scanner lectureClavier = new Scanner(System.in);
    System.out.print(" Position en x :  ");
    x = lectureClavier.nextInt();
```

Copyright © 2016 Eyrolles.

```
    System.out.print(" Position en y :  ");
    y = lectureClavier.nextInt();
    do  {
      System.out.print(" Rayon           :  ");
      r = lectureClavier.nextInt();
    } while ( r < 0 || r > TailleEcran) ;
}

public void agrandir(int nr)  {
    if (r + nr < 0) r = 0;
    else if ( r + nr > TailleEcran) r = TailleEcran ;
    else r = r + nr;
 }
```

Des méthodes invisibles

Comme nous l'avons observé précédemment, les données d'une classe sont généralement déclarées en mode `private`. Les méthodes, quant à elles, sont le plus souvent déclarées `public`, car ce sont elles qui permettent l'accès aux données protégées. Dans certains cas particuliers, il peut arriver que certaines méthodes soient définies en mode `private` . Elles deviennent alors inaccessibles depuis les classes extérieures.

Ainsi, le contrôle systématique des données est toujours réalisé par l'objet lui-même, et non par l'application qui utilise les objets. Par conséquent, les méthodes qui ont pour charge de réaliser cette vérification peuvent être définies comme méthodes internes à la classe puisqu'elles ne sont jamais appelées par l'application.

Par exemple, le contrôle de la validité de la valeur du rayon n'est pas réalisée par l'application `FaireDesCercles` mais correspond à une opération interne à la classe `Cercle`. Ce contrôle est réalisé différemment suivant que le cercle est à créer ou à agrandir (voir les méthodes `créer()` et `agrandir()` ci-dessus).

- Soit le rayon n'est pas encore connu, et la vérification s'effectue dès la saisie de la valeur. C'est ce que réalise la méthode suivante :

```
private int rayonVérifié() {
 int tmp;
 do  {
    System.out.print(" Rayon           :  ");
     tmp = lectureClavier.nextInt();
  } while ( tmp < 0 || tmp > TailleEcran) ;
return tmp;
}
```

Copyright © 2016 Eyrolles.

- Soit le rayon est déjà connu, auquel cas la vérification est réalisée à partir de la valeur passée en paramètre de la méthode :

```
private int rayonVérifié (int tmp) {
 if (tmp < 0) return 0;
   else if ( tmp > TailleEcran) return TailleEcran ;
   else return tmp;
}
```

Remarque

Les méthodes `rayonVérifié()` sont appelées **méthodes d'implémentation** ou encore méthodes « métier », car elles sont déclarées en mode privé. Leur existence n'est connue d'aucune autre classe. Seules les méthodes de la classe `Cercle` peuvent les exploiter, et elles ne sont pas directement exécutables par l'application. Elle sont cependant très utiles à l'intérieur de la classe où elles sont définies (voir les sections « Les constructeurs » et « L'héritage »).

Notons que nous venons de définir deux méthodes portant le nom `rayonVérifié()`. Le langage Java n'interdit pas la définition de méthodes portant le même nom.

Remarque

Dans cette situation, on dit que ces méthodes sont **surchargées** (voir la section « La surcharge de constructeurs »).

Les constructeurs

Grâce aux différents niveaux de protection et aux méthodes contrôlant l'accès aux données, il devient possible de construire des outils appropriés aux objets manipulés.

Cependant, la protection des données d'une classe passe aussi par la notion de constructeurs d'objets. En effet, les constructeurs sont utilisés pour initialiser correctement les données d'un objet au moment de la création de l'objet en mémoire.

Le constructeur par défaut

Le langage Java définit, pour chaque classe construite par le programmeur, un constructeur par défaut. Celui-ci initialise, lors de la création d'un objet, toutes les données de cet objet à `0` pour les entiers, à `0.0` pour les réels, à `'\0'` pour les caractères et à `null` pour les `String` ou autres types structurés.

Le constructeur par défaut est appelé par l'opérateur `new` lors de la réservation de l'espace mémoire. Ainsi, lorsque nous écrivons :

```
Cercle C = new Cercle();
```

Copyright © 2016 Eyrolles.

nous utilisons le terme `Cercle()`, qui représente en réalité le constructeur par défaut (il ne possède pas de paramètre) de la classe `Cercle`.

Un constructeur est une méthode, puisqu'il y a des parenthèses `()` derrière son nom d'appel, qui porte le nom de la classe associée au type de l'objet déclaré.

Définir le constructeur d'une classe

L'utilisation du constructeur par défaut permet d'initialiser systématiquement les données d'une classe. L'initialisation proposée peut parfois ne pas être conforme aux valeurs demandées par le type.

Dans ce cas, le langage Java offre la possibilité de définir un constructeur propre à la classe de l'objet utilisé. Cette définition est réalisée en écrivant une méthode portant le même nom que sa classe. Les instructions qui la composent permettent d'initialiser les données de la classe, conformément aux valeurs demandées par le type choisi.

Par exemple, le constructeur de la classe `Cercle` peut s'écrire de la façon suivante :

```
public Cercle()  {
   Scanner lectureClavier = new Scanner(System.in);
   System.out.print(" Position en x :  ");
   x = lectureClavier.nextInt();
   System.out.print(" Position en y :  ");
   y = lectureClavier.nextInt();
   r = rayonVérifié();
}
```

En observant la structure du constructeur `Cercle()`, nous constatons qu'un constructeur n'est pas typé. Aucun type de retour n'est placé dans son en-tête.

Question Que se passe-t-il si l'on écrit l'en-tête du constructeur comme suit :

```
public void Cercle()
```

ou encore :

```
public int Cercle()
```

Réponse Le fait de placer un type (`int`, `void`, ...) dans l'en-tête du constructeur a pour conséquence de créer une méthode, qui a pour nom `Cercle()`. Il s'agit bien d'une méthode et non d'un constructeur. Elle n'est donc pas appelée par l'opérateur `new`.

Une fois correctement défini, le constructeur est appelé par l'opérateur `new`, comme pour le constructeur par défaut. L'instruction :

```
Cercle A = new Cercle();
```

Copyright © 2016 Eyrolles.

fait appel au constructeur défini ci-dessus. Le programme exécuté demande, dès la création de l'objet `A`, de saisir les données le concernant, avec une vérification concernant la valeur du rayon grâce à la méthode `rayonVérifié()`. De cette façon, l'application est sûre d'exploiter des objets dont la valeur est valide dès leur initialisation.

Remarques

Lorsqu'un constructeur est défini par le programmeur, le constructeur proposé par défaut par le langage Java n'existe plus.

La méthode `créer()` et le constructeur ainsi définis ont un rôle identique. La méthode `créer()` devient par conséquent inutile.

La surcharge de constructeurs

Le langage Java permet la définition de plusieurs constructeurs, ou méthodes, à l'intérieur d'une même classe, du fait que la construction des objets peut se réaliser de différentes façons. Lorsqu'il existe plusieurs constructeurs, on dit que le constructeur est **surchargé**.

Dans la classe `Cercle`, il est possible de définir deux constructeurs supplémentaires :

```
public Cercle(int centrex, int centrey)    {
  x = centrex ;
  y = centrey;
}

public Cercle(int centrex, int centrey, int rayon)    {
  this( centrex, centrey) ;
  r = rayonVérifié(rayon);
}
```

Pour déterminer quel constructeur doit être utilisé, l'interpréteur Java regarde, lors de son appel, la liste des paramètres définis dans chaque constructeur. La construction des trois objets `A`, `B` et `C` suivants fait appel aux trois constructeurs définis précédemment :

```
Cercle A = new Cercle();
Cercle B = new Cercle(10, 10);
Cercle C = new Cercle(10, 10, 30);
```

Lors de la déclaration de l'objet `A`, le constructeur appelé est celui qui ne possède pas de paramètre (le constructeur par défaut, défini à la section « Définir le constructeur d'une classe »), et les valeurs du centre et du rayon du cercle `A` sont celles saisies au clavier par l'utilisateur.

La création de l'objet `B` fait appel au constructeur qui possède deux paramètres de type entier. Les valeurs du centre du cercle `B` sont donc celles passées en paramètre du constructeur, soit `(10, 10)` pour `(B.x, B.y)`. Aucune valeur n'étant précisée pour le rayon, `B.r` est automatiquement initialisé à `0`.

Copyright © 2016 Eyrolles.

Le mot-clé this

La création de l'objet `C` est réalisée par le constructeur qui possède trois paramètres entiers. Ces paramètres permettent l'initialisation de toutes les données définies dans la classe `Cercle`.

Signalons que, grâce à l'instruction `this(centrex, centrey)`, le constructeur possédant deux paramètres est appelé à l'intérieur du constructeur possédant trois paramètres.

Le mot-clé `this()` représente l'appel au second constructeur de la même classe possédant deux paramètres entiers, puisque `this()` est appelé avec deux paramètres entiers. Il permet l'utilisation du constructeur précédent pour initialiser les coordonnées du centre avant d'initialiser correctement la valeur du rayon grâce à la méthode `rayonVérifié(rayon)`, qui est elle-même surchargée. Comme pour les constructeurs, le compilateur choisit la méthode `rayonVérifié()`, définie avec un paramètre entier.

Pour finir, notons que le terme `this()` doit toujours être placé comme première instruction du constructeur qui l'utilise.

L'héritage

L'héritage est le dernier concept fondamental de la programmation objet étudiée dans ce chapitre. Ce concept permet la réutilisation des fonctionnalités d'une classe, tout en apportant certaines variations, spécifiques de l'objet héritant.

Avec l'héritage, les méthodes définies pour un ensemble de données sont réutilisables pour des variantes de cet ensemble. Par exemple, si nous supposons qu'une classe `Forme` définisse un ensemble de comportements propres à toute forme géométrique, alors :

- Ces comportements peuvent être réutilisés par la classe `Cercle`, qui est une forme géométrique particulière. Cette réutilisation est effectuée sans avoir à modifier les instructions de la classe `Forme`.
- Il est possible d'ajouter d'autres comportements spécifiques des objets `Cercle`. Ces nouveaux comportements sont valides uniquement pour la classe `Cercle` et non pour la classe `Forme`.

La relation « est un »

En pratique, pour déterminer si une classe `B` **hérite** d'une classe `A`, il suffit de savoir s'il existe une relation « **est un** » entre `B` et `A`. Si tel est le cas, la syntaxe de déclaration est la suivante :

```
class B extends A    {
// données et méthodes de la classe B
}
```

Copyright © 2016 Eyrolles.

Dans ce cas, on dit que :

- B est une **sous-classe** de A ou encore une **classe dérivée** de A.
- A est une **super-classe** ou encore une **classe de base**.

Un cercle « est une » forme géométrique

En supposant que la classe Forme possède des caractéristiques communes à chaque type de forme géométrique (les coordonnées d'affichage à l'écran, la couleur, etc.), ainsi que des comportements communs (afficher, déplacer, etc.), la classe Forme s'écrit de la façon suivante :

```
import java.util.*;
public class Forme    {
 protected int x, y ;
 private couleur ;

 public  Forme() {       // Le constructeur de la classe Forme
   Scanner lectureClavier = new Scanner(System.in);
   System.out.print(" Position en x :  ");
   x = lectureClavier.nextInt();
   System.out.print(" Position en y :  ");
   y = lectureClavier.nextInt();
   System.out.print(" Couleur de la forme :  ");
   couleur = lectureClavier.nextInt();
 }

 public void afficher()  { //Affichage des données de la classe
  System.out.println(" Position en " + x + ", " + y);
  System.out.println(" Couleur : " + couleur);
 }

 public void déplacer(int nx, int ny) {  // Déplace les coordonnées
  x = nx;                    // de la forme en (nx, ny) passées en
  y = ny;                    // paramètre de la fonction
 }
} // Fin de la classe Forme
```

Sachant qu'un objet Cercle « est une » forme géométrique particulière, la classe Cercle hérite de la classe Forme en écrivant :

```
public class Cercle extends Forme   {
 private int r ;      // rayon
 public Cercle()  {  // Le constructeur de la classe Cercle
```

Copyright © 2016 Eyrolles.

```
    System.out.print(" Rayon              :  ");
    r = rayonVérifié();
  }
  private int rayonVérifié() {
   // Voir la section "Des méthodes invisibles"
  }
  private int rayonVérifié (int tmp) {
   // Voir la section "Des méthodes invisibles"
  }
  public void afficher() { // Affichage des données de la classe
    super.afficher();
    System.out.println(" Rayon : " + r);
  }
  public double périmètre() {
   // voir la section "La classe descriptive du type Cercle" du
   // chapitre "Les classes et les objets"
  }
  public void agrandir(int nr) { // Augmente la valeur courante du
   r = rayonVérifié(r + nr);     // rayon avec la valeur passée en
  }                              // paramètre
} // Fin de la classe Cercle
```

Un cercle est une forme géométrique (`Cercle extends Forme`) qui possède un rayon (`private int r`) et des comportements propres aux cercles, soit par exemple, le calcul du périmètre (`périmètre()`) ou encore la modification de sa taille (`agrandir()`). Un cercle peut être déplacé, comme toute forme géométrique. Les méthodes de la classe `Forme` restent donc opérationnelles pour les objets `Cercle`.

En examinant de plus près les classes `Cercle` et `Forme`, nous observons que :

- La notion de constructeur existe aussi pour les classes dérivées (voir la section « Le constructeur d'une classe héritée »).
- Les données `x`, `y` sont déclarées `protected` (voir la section « La protection des données héritées »).
- La fonction `afficher()` existe sous deux formes différentes dans la classe `Forme` et la classe `Cercle`. Il s'agit là du concept de polymorphisme (voir la section « Le polymorphisme »).

Le constructeur d'une classe héritée

Les classes dérivées possèdent leurs propres constructeurs qui sont appelés par l'opérateur `new`, comme dans :

```
Cercle A = new Cercle( );
```

Copyright © 2016 Eyrolles.

Pour construire un objet dérivé, il est indispensable de construire d'abord l'objet associé à la classe mère. Pour construire un objet `Cercle`, nous devons définir ses coordonnées et sa couleur. Le constructeur de la classe `Cercle` doit appeler le constructeur de la classe `Forme`.

Remarque

Par défaut, s'il n'y a pas d'appel explicite au constructeur de la classe supérieure, comme c'est le cas dans notre exemple, le compilateur recherche de lui-même le constructeur par défaut (sans paramètre) de la classe supérieure.

En construisant l'objet `A`, l'interpréteur exécute aussi le constructeur par défaut de la classe `Forme`. L'ensemble des données du cercle (`x`, `y`, `couleur` et `r`) est alors correctement initialisé par saisie des valeurs au clavier.

Question

Que se passe-t-il si nous remplaçons le constructeur de la classe `Forme` par :

```
public Forme(int nx, int ny) {     // Le nouveau constructeur de la
  x = nx ;                         // classe Forme
  y = ny ;
  couleur = 0;
}
```

Réponse

Lors de la construction de l'objet `A`, le compilateur recherche le constructeur par défaut de la classe supérieure, soit `Forme()` sans paramètre. Ne le trouvant pas, il annonce une erreur du type `no constructor matching Forme() found in class Forme`.

Lorsqu'il n'existe pas de constructeur par défaut dans la classe supérieure, l'interpréteur ne peut plus l'exécuter au moment de la construction de l'objet de la classe dérivée. Il est nécessaire de faire appel à l'outil `super()`.

Le mot-clé super

Pour éviter ce type d'erreur, la solution consiste à appeler directement le constructeur de la classe mère depuis le constructeur de la classe :

```
// Le constructeur de la classe Cercle
public Cercle(int xx, int yy)  {
   super(xx, yy);
   System.out.print(" Rayon            :  ");
   r = rayonVérifié();
}
```

De cette façon, le terme `super()` représentant le constructeur de la classe supérieure possédant deux entiers en paramètres, l'interpréteur peut finalement construire l'objet `A`

Copyright © 2016 Eyrolles.

(`Cercle A = new Cercle(5, 5)`), par appel du constructeur de la classe `Forme` à l'intérieur du constructeur de la classe `Cercle`.

Remarque Le terme `super` est obligatoirement la première instruction du constructeur de la classe dérivée. La liste des paramètres (deux `int`) permet de préciser au compilateur quel est le constructeur utilisé en cas de surcharge de constructeurs.

La protection des données héritées

En héritant de la classe `A`, la classe `B` hérite des données et méthodes de la classe `A`. Cela ne veut pas forcément dire que la classe `B` ait accès à toutes les données et méthodes de la classe `A`. En effet, héritage n'est pas synonyme d'accessibilité.

Remarque Lorsqu'une donnée de la classe supérieure est déclarée en mode `private`, la classe dérivée ne peut ni consulter ni modifier directement cette donnée héritée. L'accès ne peut se réaliser qu'au travers des méthodes de la classe supérieure.

Pour notre exemple, la donnée `couleur` étant déclarée `private` dans la classe `Forme`, le constructeur suivant génère l'erreur `variable couleur in class Forme not accessible from class Cercle`.

```
public Cercle(int xx, int yy)    {  // Le constructeur de la classe
   super(xx, yy);                   // Cercle
   couleur = 20 ;
   r = 10;
}
```

Il est possible, grâce à la protection `protected`, d'autoriser l'accès en consultation et modification des données de la classe supérieure. Toutes les données de la classe `A` sont alors accessibles depuis la classe `B`, mais pas depuis une autre classe.

Dans notre exemple, si la donnée `couleur` est déclarée `protected` dans la classe `Forme`, alors le constructeur de la classe `Cercle` peut modifier sa valeur.

Le polymorphisme

La notion de polymorphisme découle directement de l'héritage. Par polymorphisme, il faut comprendre qu'une méthode peut se comporter différemment suivant l'objet sur lequel elle est appliquée.

Lorsqu'une même méthode est définie à la fois dans la classe mère et dans la classe fille, l'exécution de la forme (méthode) choisie est réalisée en fonction de l'objet associé à l'appel

Copyright © 2016 Eyrolles.

et non plus suivant le nombre et le type des paramètres, comme c'est le cas lors de la surcharge de méthodes à l'intérieur d'une même classe.

Pour notre exemple, la méthode `afficher()` est décrite dans la classe `Forme` et dans la classe `Cercle`. Cette double définition ne correspond pas à une véritable surcharge de fonctions. Ici, les deux méthodes `afficher()` sont définies sans aucun paramètre. Le choix de la méthode ne peut donc s'effectuer sur la différence des paramètres. Il est effectué par rapport à l'objet sur lequel la méthode est appliquée. Observons l'exécution du programme suivant :

```
public class FormerDesCercles    {
 public static void main(String [] arg)  {
  Cercle A  = new Cercle(5, 5);
  A.afficher();
  Forme F = new Forme (10, 10, 3);
  F.afficher();
 }
}
```

L'appel du constructeur de l'objet `A` nous demande de saisir la valeur du rayon :

```
Rayon              : 7
```

La méthode `afficher()` est appliquée à `A`. Puisque `A` est de type `Cercle`, l'affichage correspond à celui réalisé par la méthode définie dans la classe `Cercle`, soit :

```
Position en 5, 5
Couleur : 20
Rayon : 7
```

La forme `F` est ensuite créée puis affichée à l'aide la méthode `afficher()` de la classe `Forme`, `F` étant de type `Forme` :

```
Position en 10, 10
Couleur : 3
```

Remarque

Lorsqu'une méthode héritée est définie une deuxième fois dans la classe dérivée, l'héritage est supprimé. Le fait d'écrire `A.afficher()` ne permet plus d'appeler directement la méthode `afficher()` de la classe `Forme`.

Pour appeler la méthode définie dans la classe supérieure, la solution consiste à utiliser le terme `super`, qui recherche la méthode à exécuter en remontant dans la hiérarchie.

Dans notre exemple, `super.afficher()` permet d'appeler la méthode `afficher()` de la classe `Forme`.

Grâce à cette technique, si la méthode d'affichage pour une `Forme` est transformée, cette transformation est automatiquement répercutée pour un `Cercle`.

Copyright © 2016 Eyrolles.

Les interfaces

Nous l'avons vu à la section « Les objets contrôlent leur fonctionnement » de ce chapitre, les objets de la vie réelle sont manipulés par une interface appropriée et les objets informatiques proposent également une interface qui nous permet de communiquer avec eux.

Qu'est-ce qu'une interface ?

En pratique, une interface correspond par exemple à une fenêtre de contrôle ou un panneau précisant les informations en cours de traitement. L'interface de communication se doit d'être suffisamment générale pour être utilisable dans le plus grand nombre de cas, tout en proposant un modèle d'utilisation approprié à sa fonction.

Une interface correspond donc à une classe qui définit non pas un modèle d'objet (une sorte de moule) mais un ensemble de comportements possibles, sans que ces comportements soient réellement décrits.

Plus précisément, lorsqu'un utilisateur clique par exemple sur le bouton « Valider » d'une application, une action doit être menée. Cette action diffère selon l'application. Il peut s'agir de confirmer l'envoi d'un message électronique ou encore de supprimer un fichier.

La classe modélisant le bouton de validation doit donc proposer une méthode nommée par exemple `actionArealiser()` (en anglais `actionPerformed()`) sans décrire explicitement le code de cette action. Seul l'utilisateur (c'est-à-dire le programmeur d'applications) est à même de décrire l'action à réaliser.

Remarque

Le traitement des événements et la description des actions associées sont étudiés plus précisément au chapitre 11 « Dessiner des objets », section « Exemple : associer un bouton à une action ».

Cette classe de modélisation des comportements correspond en pratique à une interface.

Syntaxe d'une interface

Une interface, dans le langage Java, définit les noms des méthodes associées aux comportements. Elle ressemble à une classe puisqu'elle s'écrit comme suit :

```
interface uneInterface {
    public type methode1() ;
    public type methode2() ;
    // d'autres en-têtes de méthodes
}
```

Copyright © 2016 Eyrolles.

Les règles d'écriture d'une interface sont simples :

- une interface est définie au sein d'un fichier qui porte son nom suivi de l'extension `.java` ;
- le terme `interface` remplace le terme `class` ;
- les comportements proposés par l'interface sont définis à partir des en-têtes de méthodes (signature).

Principe de fonctionnements

Une fois l'interface définie, les méthodes sont concrètement décrites au sein des différentes classes qui implémentent l'interface. Pour cela, vous devez :

- placer le terme `implements` lors de la création de la classe ;
- décrire ce que réalise les méthodes, comme suit :

```
public class UneClasse implements uneInterface {
    public type methode1() {
    // Ici sont décrites les actions à mener pour cette méthode au sein de cette classe
    }
    public type methode2() {
    // Ici sont décrites les actions à mener pour cette méthode au sein de cette classe
    }
}
```

Une seconde classe peut également implémenter la même interface et dans ce cas, le code s'écrit :

```
public class UneAutreClasse implements uneInterface {
    public type methode1() {
    // Ici sont décrites les actions à mener pour cette méthode au sein de cette classe
    }
    public type methode2() {
    // Ici sont décrites les actions à mener pour cette méthode au sein de cette classe
    }
}
```

Les méthodes `methode1()` et `methode2()` ne se comportent pas de la même façon selon la classe dans laquelle elles sont définies. Elles contiennent des instructions différentes d'une classe à l'autre.

Les objets implémentant l'interface `uneInterface` sont ensuite créés dans l'application de la façon suivante :

```
UneClasse premier = new UneClasse();
UneAutreClasse second = new UneAutreClasse();
// appeler la methode1() de UneClasse
```

Copyright © 2016 Eyrolles.

```
premier.methode1();
// appeler la methode1() de UneAutreClasse
second.methode1();
```

Remarque Une interface modélise des comportements, et non des objets. Il est donc impossible de créer une instance d'interface à l'aide de l'opérateur `new`.

Avec la version 8 de Java, il est possible de créer des méthodes par défaut au sein d'une interface. Pour cela, il suffit d'ajouter le terme `default` devant l'en-tête de la méthode comme suit :

```
interface uneInterface {
   default public type methode1() {
     System.out.print("Je suis une méthode par défaut ! ") ;
   };
   public type methode2() ;
   // d'autres en-têtes de méthodes
}
```

Utiliser les méthodes par défaut a pour avantage de permettre l'écriture d'un code réutilisable dans le temps surtout lors de l'ajout de nouvelles fonctionnalités. En effet, il n'est pas nécessaire de redéfinir une méthode par défaut au sein d'une classe qui implémente une interface. Vous pouvez ainsi utiliser la méthode par défaut qui fonctionne comme vous le souhaitez. Et, dans le cas d'un ajout de fonctionnalités, vous pouvez modifier la méthode par défaut au sein d'une nouvelle classe, sans pour autant affecter le bon fonctionnement du reste de votre application.

Calculs géométriques

L'objectif de cet exemple est de construire une interface qui permette de calculer n'importe quels surface et périmètre d'une forme géométrique. Pour cela, nous allons utiliser les notions d'héritage et d'interface étudiées au cours des deux sections précédentes.

Cahier des charges

L'application principale crée autant de formes qu'elle le souhaite (cercle ou rectangle) en utilisant les classes `Forme`, `Cercle` et `Rectangle`.

Pour en savoir plus La classe `Rectangle` est réalisée en exercice, à la fin de ce chapitre.

La classe `Forme` implémente l'interface `CalculGeometrique` qui définit deux méthodes `surface()` et `perimetre()`.

Copyright © 2016 Eyrolles.

Le calcul de la surface d'une forme au sein de la classe `Forme` n'est pas possible, car la forme n'est pas encore réellement définie. Les méthodes `surface()` et `perimetre()` au sein de cette classe retournent une valeur négative.

Les classes `Cercle` et `Rectangle` héritent toutes deux de la classe `Forme`, elles implémentent par conséquent l'interface `CalculGeometrique`. Les méthodes `surface()` et `perimetre()` au sein de ces différentes classes retournent la valeur du périmètre et de la surface calculée à partir de la formule correspondant à la forme géométrique associée à la classe.

Exemple : code source

L'interface `CalculGeometrique` définit les méthodes `surface()` et `perimetre()` comme suit :

```
interface CalculGeometrique {
     public double surface();
     public double perimetre();
}
```

La classe `Forme` implémente l'interface `CalculGeometrique` et décrit explicitement les méthodes `surface()` et `perimetre()`. Ces dernières retournent la valeur `-1`.

```
public class Forme implements CalculGeometrique {
   protected int x, y, couleur ;
   // Définition du constructeur et de la méthode afficher()
   // Description des méthodes surface() et perimetre() pour la classe Forme
   public double surface() {
          return -1 ;
   }
   public double perimetre() {
          return -1 ;
   }
```

La classe `Cercle` hérite de la classe `Forme`. Par héritage, elle implémente l'interface `CalculGeometrique`. Les méthodes `perimetre()` et `surface()` associées à la classe `Cercle` retournent respectivement la valeur correspondant au calcul mathématique du périmètre d'un cercle et de sa surface.

```
public class Cercle extends Forme{
   private int r ;
   public double surface() {
     return Math.PI *r*r ;
   }
   public double perimetre() {
      return 2*Math.PI*r ;
   }
}
```

Copyright © 2016 Eyrolles.

L'application finale crée des objets de type `Cercle` ou `Forme` comme suit :

```
Cercle A  = new Cercle(5, 5);
A.afficher();
if ( A.perimetre() >=0) {
  System.out.println("Le périmètre de A vaut " + A.perimetre());
} else {
  System.out.println("Calcul impossible");
}
Forme F = new Forme (10, 10);
F.afficher();
if ( F.perimetre() >=0) {
  System.out.println("Le périmètre de F vaut " + F.perimetre());
} else {
  System.out.println("Calcul impossible");
}
```

Le périmètre d'un cercle ou d'une forme est calculé en utilisant la méthode correspondant au type de l'objet utilisé.

Exemple avec les méthodes par défaut

L'interface `CalculGeometrique` définit les méthodes par défaut `surface()` et `perimetre()` comme suit :

```
interface CalculGeometrique {
   default public double surface() {
        return -1;
   }
   default public double perimetre() {
        return -1;
   }
   public double autremethode();
}
```

La classe `Forme` implémente l'interface `CalculGeometrique` qui n'a plus besoin de décrire explicitement les méthodes `surface()` et `perimetre()`. Elle utilise les comportements par défaut de l'interface `CalculGeometrique`.

```
public class Forme implements CalculGeometrique {
   protected int x, y, couleur ;
   // Définition du constructeur et de la méthode afficher()
   // Description des méthodes afficher et echangerAvec pour la classe Forme
}
```

Copyright © 2016 Eyrolles.

La classe `Cercle` hérite de la classe `Forme`. Par héritage, elle implémente l'interface `CalculGeometrique`.

```
public class Cercle extends Forme{
   private int r ;
   public double surface() {
     return Math.PI *r*r ;
   }
   public double perimetre() {
      return 2*Math.PI*r ;
   }
}
```

Les méthodes `perimetre()` et `surface()` associées à la classe `Cercle` ont un autre comportement que celui par défaut. En les redéfinissant au sein de la classe `Cercle`, l'appel de la méthode `perimetre()` ou `surface()` par l'intermédiaire d'un objet de type `Cercle` aura pour résultat de retourner le périmètre ou la surface de l'objet et non la valeur -1.

Résumé

Lorsque l'interpréteur Java rencontre le mot-clé `static` devant une variable (**variable de classe**), il réserve un seul et unique emplacement mémoire pour cette variable. Si ce mot-clé est absent, l'interpréteur peut construire en mémoire la variable déclarée non `static` (**variable d'instance**) en plusieurs exemplaires. Cette présence ou cette absence du mot-clé `static` permet de différencier les variables des objets.

Les objets sont définis en mémoire par l'intermédiaire d'une **adresse** (**référence**). Lorsqu'un objet est passé en paramètre d'une fonction, la valeur passée au paramètre formel est l'adresse de l'objet. De cette façon, si la méthode transforme les données du paramètre formel, elle modifie aussi les données de l'objet effectivement passé en paramètre. Ainsi, tout objet passé en paramètre d'une méthode voit, en sortie de la méthode, ses données transformées par la méthode. Ce mode de transmission des données est appelé **passage** de paramètres **par référence**.

L'objectif principal de la programmation objet est d'écrire des programmes qui contrôlent par eux-mêmes le bien-fondé des opérations qui leur sont appliquées. Ce contrôle est réalisé grâce au principe d'**encapsulation** des données. Par ce terme, il faut comprendre que les données d'un objet sont protégées, de la même façon qu'un médicament est protégé par la fine capsule qui l'entoure. L'encapsulation passe par le **contrôle des données** et des comportements de l'objet à travers les niveaux de **protection**, l'**accès** contrôlé aux données et la notion de **constructeur** de classe.

Le langage Java propose trois niveaux de protection, `public`, `private` et `protected`. Lorsqu'une donnée est totalement protégée (`private`), elle ne peut être modifiée que par les méthodes de la classe où la donnée est définie.

Copyright © 2016 Eyrolles.

On distingue les méthodes qui consultent la valeur d'une donnée sans pouvoir la modifier (**accesseur en consultation**) et celles qui modifient après contrôle et validation la valeur de la donnée (**accesseur en modification**).

Les constructeurs sont des méthodes particulières, déclarées uniquement `public`, qui portent le même nom que la classe où ils sont définis. Ils permettent le contrôle et la validation des données dès leur initialisation.

Par défaut, si aucun constructeur n'est défini dans une classe, le langage Java propose un constructeur par défaut, qui initialise toutes les données de la classe à `0` ou à `null`, si les données sont des objets. Si un constructeur est défini, le constructeur par défaut n'existe plus.

L'**héritage** permet la réutilisation des objets et de leur comportement, tout en apportant de légères variations. Il se traduit par le principe suivant : on dit qu'une classe `B` hérite d'une classe `A` (`B` étant une sous-classe de `A`) lorsqu'il est possible de mettre la relation « est un » entre `B` et `A`.

De cette façon, toutes les méthodes, ainsi que les données déclarées `public` ou `protected`, de la classe `A` sont applicables à la classe `B`. La syntaxe de déclaration d'une sous-classe est la suivante :

```
class B extends A    {
// données et méthodes de la classe B
}
```

Le terme `implements` est utilisé pour créer des interfaces. Une interface définit tous les types de comportements d'un objet sans décrire explicitement le code. Avec la version 8 de Java, il est possible de définir des méthodes par défaut au sein d'une interface, grâce au terme `default`.

Exercices

La protection des données

Les méthodes d'accès en écriture

Exercice 8.1 Reprendre l'application `Bibliotheque` et la classe `Livre` développées au cours de l'exercice 7.4 du chapitre précédent et, modifier les propriétés de la classe, de façon à les rendre privées. Que se passe-t-il lors de la compilation de l'application `Bibliotheque` ? Pourquoi ?

Exercice 8.2 Pour corriger l'erreur de compilation, vous devez mettre en place des méthodes d'accès en écriture afin de permettre la modification des propriétés depuis l'extérieur de la classe `Livre`.

a. En supposant que la méthode `setTitre()` est appelée depuis la fonction `main()` comme suit :

```
System.out.print("Entrez le titre : ");
livrePoche.setTitre(lectureClavier.next());
```

insérer à l'intérieur de la classe `Livre` la méthode `setTitre()` afin de pouvoir modifier la propriété `titre`.

Copyright © 2016 Eyrolles.

b. En vous inspirant de la méthode `setTitre()`, créez les méthodes autorisant la modification des autres propriétés de la classe indépendamment les unes des autres.

c. Modifier l'application `Bibliotheque` en tenant compte des nouvelles méthodes d'accès en écriture.

d. Est-il nécessaire de créer une méthode `setCode()` ? Pourquoi ?

Les méthodes d'accès en lecture

Exercice 8.3 a. Pour faire en sorte que l'application `Bibliotheque` puisse afficher les propriétés de la classe `Livre` indépendamment les unes des autres, insérer à l'intérieur de la classe `Livre` les méthodes `getTitre()`, `getNomAuteur()`, `getPrenomAuteur()`, `getCategorie()`, `getIsbn()` et `getCode()`. Ces méthodes retournent au programme appelant la propriété indiquée par le nom de la méthode.

b. Modifier l'application `Bibliotheque` afin de n'afficher que le titre et le code du livre `livrePoche`.

Les méthodes invisibles (métier)

Exercice 8.4 Pour répondre à la question 8.2.d, renommez la méthode `calculerLeCode()` par `setCode()` et faites en sorte que cette méthode ne soit pas accessible par aucune autre classe que la classe `Livre`.

Les constructeurs

Exercice 8.5 a. Le constructeur par défaut de la classe `Livre` permet de saisir les données d'un livre. Écrire le constructeur en utilisant les méthodes d'accès en écriture réalisées en 8.2.

Remarque L'utilisation des méthodes d'accès en écriture au sein du constructeur `Livre` n'est pas réellement obligatoire. Mais plus généralement, cela peut être utile pour s'assurer de la validité des données (voir exercice 8.6, ci-après).

b. Sans la modifier, exécutez l'application `Bibliotheque`. Que se passe-t-il ? Pourquoi ? Transformez l'application afin d'éviter ce problème.

c. Surchargez le constructeur par défaut, en définissant un nouveau constructeur qui initialise les propriétés d'un livre à partir des valeurs qui lui sont fournis en paramètre.

Copyright © 2016 Eyrolles.

d. Dans l'application `Bibliotheque`, créez un objet `unPolar` initialisé dès la création aux données suivantes : `"Le mystère de la chambre jaune"`, `"Leroux"`, `"Gaston"`, `"Policier"` et `"2253005495"`. Affichez le contenu de l'objet `unPolar`.

L'héritage

En examinant la figure 8.6, et en vous aidant des notions acquises au cours des exercices précédents et des exercices réalisés au chapitre 7, nous allons créer les classes `Cercle`, `Rectangle` et `Triangle` à partir de la classe `Forme`.

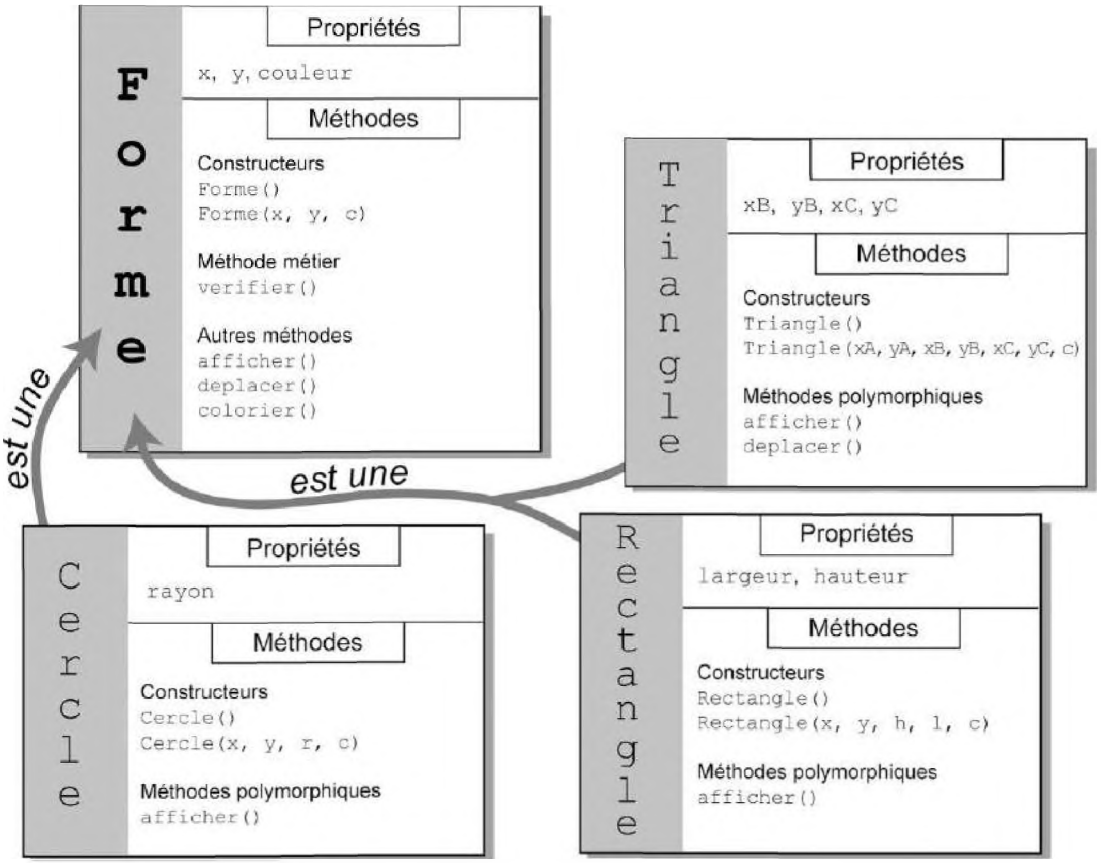


Figure 8-6 *Le cercle, le rectangle et le triangle sont des formes. Les classes qui les définissent héritent de la classe mère Forme.*

Copyright © 2016 Eyrolles.

La classe Forme

Exercice 8.6 Sachant que toute forme géométrique est définie par :

- Une couleur.
- Une position en X et en Y définissant les coordonnées du point de référence pour placer la forme à l'écran.

Et que :

- La couleur varie entre `0` et `10`.
- La propriété `X` est comprise entre `0` et `800`.
- La propriété `Y` est comprise entre `0` et `600`.

a. Définir les propriétés de la classe `Forme` en mode `protected`.

b. Définir des constantes pour la largeur (`800`) et la hauteur (`600`) de la fenêtre d'affichage ainsi que pour le nombre de couleurs maximum proposé (`10`).

c. Reprendre la méthode `verifier()` de l'exercice 7.7 du chapitre précédent et définissez la comme une méthode métier (invisible).

La méthode vérifie la validité des valeurs pour toutes les propriétés de la classe (`couleur`, `x`...), modifiez la méthode de façon à passer en paramètre un message indiquant à quelle propriété sera attribuée la saisie. L'appel à la méthode pourra s'effectuer de la façon suivante :

```
couleur = verifier("couleur", 0, couleurMax);
```

ou encore

```
largeur = verifier("Largeur", 0, largeurEcran);
```

Surcharger la méthode `verifier()` en créant une méthode vérifiant une valeur passée en paramètre.

d. Écrire un constructeur :

- par défaut qui permet de saisir les données d'une forme. Les données saisies doivent être vérifiées en utilisant la première forme de la méthode `verifier()` ;
- muni des trois paramètres permettant d'initialiser directement, les propriétés de la classe `Forme`. Les données passées en paramètre doivent être vérifiées en utilisant la seconde forme de la méthode `verifier()`.

e. Écrire la méthode `deplacer()` qui déplace une forme à partir des valeurs passées en paramètres. Par exemple si le point de référence de la forme est positionnée en `100, 100`, la méthode `deplacer(10, 10)` a pour résultat de placer le point de référence de la forme en `110, 110`. Les nouvelles coordonnées de la forme doivent être vérifiées.

f. Écrire la méthode `colorier()` qui change la couleur d'une forme en fonction de la valeur passée en paramètre. La valeur de la nouvelle couleur doit être vérifiée.

g. Écrire la méthode `afficher()` qui affiche les propriétés de la classe `Forme`.

Copyright © 2016 Eyrolles.

La classe Rectangle

Exercice 8.7 Sachant que tout rectangle est une forme géométrique possédant une hauteur dont la valeur est comprise entre 0 et 600, et une largeur dont la valeur est comprise entre 0 et 800 :

a. Définir la classe `Rectangle` à partir de la classe `Forme`.

b. Définir les propriétés de la classe `Rectangle` en mode privée.

c. Écrire un constructeur :

- par défaut qui permet de saisir la hauteur et la largeur d'un rectangle. Ces valeurs doivent être vérifiées ;
- muni de cinq paramètres permettant d'initialiser directement l'ensemble des propriétés `x`, `y`, `couleur`, `largeur` et `hauteur` de la classe `Rectangle`. Ce constructeur fait appel au constructeur avec paramètre, de la classe `Forme`. Les données passées en paramètres doivent être vérifiées.

d. Écrire la méthode `afficher()` qui affiche les propriétés de la classe `Rectangle` ainsi que celles de la classe `Forme`.

e. Écrire les méthodes `perimetre()` et `surface()` qui calculent le périmètre et la surface d'un rectangle.

La classe Triangle

Exercice 8.8 Sachant que tout triangle est une forme géométrique possédant trois sommets dont les valeurs en `X` sont comprises entre 0 et 800 et en en `Y` sont comprises entre 0 et 600 :

a. Définir la classe `Triangle` à partir de la classe `Forme`.

b. Définir les propriétés de la classe `Triangle` en mode privée. Les coordonnées `X` et `Y` de la classe `Forme` – point de référence du triangle – correspondent aux coordonnées du premier sommet.

c. Écrire un constructeur :

- par défaut qui permet de saisir des deux sommets restants du triangle. Ces valeurs doivent être vérifiées.
- muni de sept paramètres permettant d'initialiser directement l'ensemble des propriétés `x`, `y`, `couleur`, `x1`, `y1`, `x2` et `y2` de la classe `Triangle`. Ce constructeur fait appel au constructeur avec paramètre de la classe `Forme`. Les données passées en paramètres doivent être vérifiées.

d. Écrire la méthode `afficher()` qui affiche les propriétés de la classe.

L'application FaireDesFormesGeometriques

Exercice 8.9 a. Écrire une application qui permet la création d'un cercle, d'un rectangle, et d'un triangle.

b. Vérifier que les valeurs des propriétés de chaque classe ne peuvent être saisies en dehors des limites imposées.

Copyright © 2016 Eyrolles.

c. Afficher les valeurs de chacune des formes.

d. Déplacer toutes les formes de `10` pixels en `X` et `20` en `Y`. Que se passe-t-il pour le triangle ? Pourquoi ? Comment faire pour que le triangle se déplace correctement ?

e. Dans la classe `Triangle`, écrire la méthode `deplacer()` afin de déplacer tous les sommets du triangle à partir des valeurs passées en paramètre. La méthode fait appel à la méthode `deplacer()` de la classe supérieure afin de déplacer le premier sommet du triangle. Elle vérifie également que les nouveaux sommets ne sortent pas de la fenêtre.

f. Afficher les périmètres et les surfaces de tous les rectangles et les cercles créés au cours de l'application.

Les interfaces

Dans le cadre du développement d'un jeu de plateau, nous souhaitons simuler les déplacements du héros en fonction du moyen de transport qu'il utilise. Il dispose d'une voiture, d'un ascenseur et d'une fusée.

L'objectif est de décrire les déplacements du personnage en utilisant le mécanisme des interfaces.

Exercice 8.10

a. Définir l'interface `Deplacement` en décrivant les méthodes `deplacementEnX()`, `deplacementEnY()` et `seDeplacer()`.

b. Créer la classe `MoyenDeTransport` qui possède trois propriétés `x`, `y` et `vitesse`, ainsi qu'une constante de déplacement `vitesseInitiale` valant `10`.

c. Dans la classe `MoyenDeTransport`, définir :

- les méthodes `deplacementEnX()` et `deplacementEnY()`. Le moyen de transport n'étant pas encore défini, aucune instruction ne les compose.
- la méthode `seDeplacer()` qui appelle les deux méthodes précédentes.
- la méthode `afficher()` qui affiche à l'écran la position du héros.

d. Chaque moyen de transport ayant sa propre limitation de vitesse, écrire la méthode `limitation()` qui initialise la vitesse de l'objet à l'aide du premier paramètre et détermine si cette nouvelle vitesse dépasse la limite fixée en second paramètre. Si tel est le cas, la vitesse de l'objet devient la vitesse limite.

Exercice 8.11 Le comportement général d'un moyen de transport étant défini, vous devez maintenant décrire chaque moyen de transport au sein d'une classe spécifique. Pour cela, il convient de :

a. Décrire la classe `Voiture`. Sachant qu'avec ce moyen de transport, le héros ne peut se déplacer qu'avec une vitesse qui est jusqu'à dix fois plus rapide que la vitesse initiale, tout en ne dépassant pas plus de 150. La classe `Voiture` hérite de la classe `MoyenDeTransport`. Une voiture ne se déplace qu'à l'horizontal, seule la méthode `deplacementEnX()` est à redéfinir, en incrémentant la position en X de l'objet avec sa vitesse.

Copyright © 2016 Eyrolles.

b. Décrire la classe `Ascenseur`. Sachant qu'avec ce moyen de transport, le héros ne peut se déplacer qu'avec une vitesse qui est jusqu'à cinq fois plus rapide que la vitesse initiale, tout en ne pouvant pas dépasser 50. La classe `Ascenseur` hérite de la classe `MoyenDeTransport`. Un ascenseur ne se déplace qu'à la verticale, seule la méthode `deplacementEnY()` est à redéfinir, en incrémentant la position en Y de l'objet avec sa vitesse.

c. Décrire la classe `Fusee`. Sachant qu'avec ce moyen de transport, le héros peut se déplacer avec une vitesse qui est jusqu'à cent fois plus rapide que la vitesse initiale, tout en ne pouvant pas dépasser 1200. La classe `Fusee` hérite de la classe `MoyenDeTransport`. Une fusée se déplace à la verticale et à l'horizontale, les deux méthodes `deplacementEnY()` et `deplacementEnY()` sont à redéfinir.

Exercice 8.12 Écrire l'application `UnHerosSeDeplace` dont la fonction `main()` crée trois véhicules de type `Voiture`, `Ascenseur` et `Fusee`.

a. Déplacer les objets en utilisant la méthode `seDeplacer()`. Vérifier que chaque objet se déplace correctement en affichant leur position.

b. Avec les nouvelles fonctionnalités de Java 8, modifier l'interface `Deplacement` de façon à définir les méthodes `deplacementEnX()` et `deplacementEnY()` comme méthodes par défaut.

b. Les deux méthodes `deplacementEnX()` et `deplacementEnY()` possédant un comportement par défaut, est-il nécessaire de définir à nouveau ces deux comportements dans la classe `MoyenDeTransport` ? Modifier la classe `MoyenDeTransport` en conséquence.

Le projet : Gestion d'un compte bancaire

Encapsuler les données d'un compte bancaire

La protection privée et l'accès aux données

a. Déclarez toutes les variables d'instance des types `Compte` et `LigneComptable` en mode `private`. Que se passe-t-il lors de la phase de compilation de l'application `Projet` ?

Pour remédier à cette situation, la solution est de construire des méthodes d'accès aux données de la classe `Compte` et `LigneComptable`. Ces méthodes ont pour objectif de fournir au programme appelant la valeur de la donnée recherchée. Par exemple, la fonction `quelTypeDeCompte()` suivante fournit en retour le type du compte recherché :

```
public String quelTypeDeCompte()    {
  return typeCpte;
}
```

b. Écrivez, suivant le même modèle, toutes les méthodes d'accès aux données `val_courante`, `taux`, `numéroCpte`, etc.

Copyright © 2016 Eyrolles.

c. Modifiez l'application `Projet` et la classe `Compte` de façon à pouvoir accéder aux données `numéroCpte` de la classe `Compte` et aux valeurs de la classe `LigneComptable`.

Le contrôle des données

L'encapsulation des données permet le contrôle de la validité des données saisies pour un objet. Un compte bancaire ne peut être que de trois types : `Epargne`, `Courant` ou `Joint`. Il est donc nécessaire, au moment de la saisie du `type` du compte, de contrôler l'exactitude du type entré. La méthode `contrôleType()` suivante réalise ce contrôle :

```
private String contrôleType()  {
 char tmpc;
 String tmpS = "Courant";
 Scanner lectureClavier = new Scanner(System.in);
 do {
  System.out.print("Type du compte [Types possibles : C(ourant),
                   J(oint), E(pargne)] : ");
  tmpc = lectureClavier.next().charAt(0);
 } while ( tmpc != 'C' && tmpc != 'J' && tmpc != 'E');
  switch (tmpc) {
   case 'C' : tmpS = "Courant";
              break;
   case 'J' : tmpS = "Joint";
              break;
   case 'E' : tmpS = "Epargne";
              break;
  }
  return tmpS;
}
```

À la sortie de la fonction, nous sommes certains que le type retourné correspond aux types autorisés par le cahier des charges.

a. Dans la classe `Compte`, sachant que la valeur initiale ne peut être négative à la création d'un compte, écrivez la méthode `contrôleValinit()`.

b. Dans la classe `LigneComptable`, écrivez les méthodes `contrôleMotif()` et `contrôleMode()`, qui vérifient respectivement le motif (`Salaire`, `Loyer`, `Alimentation`, `Divers`) et le mode (`CB`, `Virement`, `Chèque`) de paiement pour une ligne comptable

Pour contrôler la validité de la date, voir la section « Le projet : Gestion d'un compte bancaire » du chapitre 10, « Collectionner un nombre indéterminé d'objets ».

Copyright © 2016 Eyrolles.

c. Modifiez les méthodes `créerCpte()` et `créerLigneComptable()` de façon à ce que les données des classes `Compte` et `LigneComptable` soient valides.

Les constructeurs de classe

Les constructeurs `Compte()` et `LigneComptable()` s'inspirent pour une grande part des méthodes `créerCpte()` et `créerLigneComptable()`.

a. Remplacez directement `créerCpte()` par `Compte()`. Que se passe-t-il lors de l'exécution du programme ?

b. Déplacez l'appel au constructeur dans l'option `1`, de façon à construire l'objet au moment de sa création. Que se passe-t-il en phase de compilation ? Pourquoi ?

c. Utilisez la notion de surcharge de constructeur pour construire un objet `C` de deux façons :

- Les valeurs initiales du compte sont passées en paramètres.
- Les valeurs initiales sont saisies au clavier, comme le fait la méthode `créerCpte()`.

d. À l'aide de ces deux constructeurs, modifiez l'application `Projet` de façon à pouvoir l'exécuter correctement.

Comprendre l'héritage

Protection des données héritées

Sachant qu'un compte d'épargne est un compte bancaire ayant un taux de rémunération :

a. Écrivez la classe `CpteEpargne` en prenant soin de déclarer la nouvelle donnée en mode `private`.

b. Modifiez le type `Compte` de façon à supprimer tout ce qui fait appel au compte d'épargne (donnée et méthodes).

Un compte d'épargne modifie la valeur courante par le calcul des intérêts, en fonction du taux d'épargne. Il ne peut ni modifier son numéro, ni son type.

c. Quels modes de protection doit-on appliquer aux différentes données héritées de la classe `Compte` ?

Le contrôle des données d'un compte d'épargne

Sachant que le taux d'un compte d'épargne ne peut être négatif, écrivez la méthode `contrôleTaux()`.

Copyright © 2016 Eyrolles.

Le constructeur d'une classe dérivée

En supposant que le constructeur de la classe `CpteEpargne` s'écrive de la façon suivante :

```
public CpteEpargne() {
  super("Epargne");
  taux = contrôleTaux();
}
```

a. Recherchez à quel constructeur de la classe `Compte` fait appel `CpteEpargne()`. Pourquoi ?

b. Modifiez ce constructeur de façon à ce que la donnée `typeCpte` prenne la valeur `Epargne`.

Le polymorphisme

De la méthode `afficherCpte()` :

a. Dans la classe `CpteEpargne`, écrivez la méthode `afficherCpte()`, sachant qu'afficher les données d'un compte d'épargne revient à afficher les données d'un compte, suivies du taux d'épargne.

De l'objet `C`, déclaré de type `Compte` :

b. Dans l'application `Projet`, modifiez l'option `1`, de façon à demander à l'utilisateur s'il souhaite créer un compte simple ou un compte d'épargne. Selon la réponse, construisez l'objet `C` en appelant le constructeur approprié.

Copyright © 2016 Eyrolles.

Copyright © 2016 Eyrolles.

Partie III

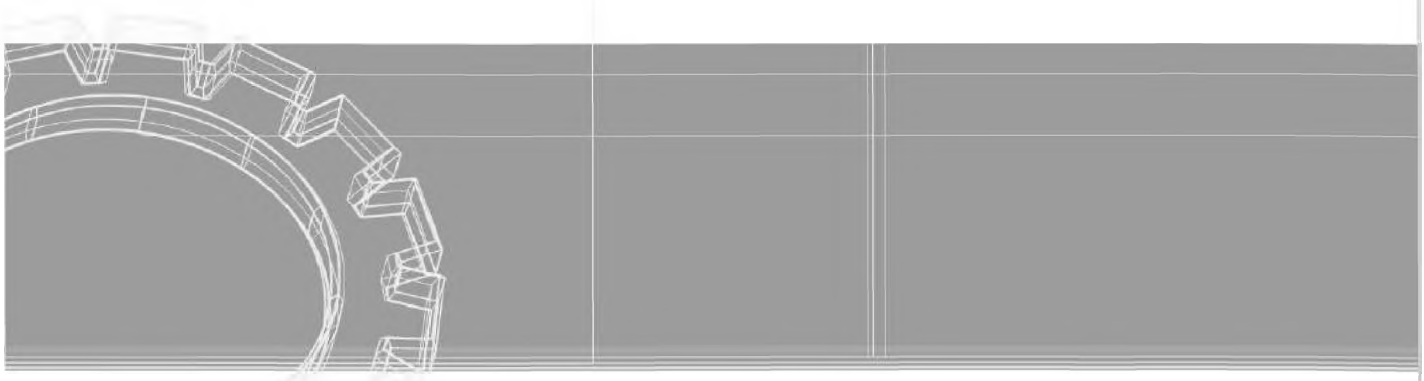
Outils et techniques orientés objet

Copyright © 2016 Eyrolles.

Copyright © 2016 Eyrolles.

Chapitre 9

Collectionner un nombre fixe d'objets



Comme nous l'avons observé tout au long de cet ouvrage, l'atout principal de l'ordinateur est sa capacité à manipuler un grand nombre de données pour en extraire de nouvelles informations. Or, les structures de stockage étudiées jusqu'ici, telles que variables ou objets, ne permettent pas d'appliquer de traitements systématiques sur des ensembles de valeurs.

C'est pourquoi nous étudions dans ce chapitre une nouvelle structure de données, les tableaux, qui permettent le stockage d'un nombre fini de valeurs.

Dans un premier temps, nous étudions « Les tableaux à une dimension » et observons comment les déclarer et les manipuler. Pour mieux comprendre la manipulation de ces structures, nous analysons ensuite, à la section « Quelques techniques utiles », différentes techniques de programmation appliquées aux tableaux à une dimension, telles que la recherche d'une valeur dans un tableau ou le tri d'un tableau.

Pour finir, nous examinons à la section « Les tableaux à deux dimensions », comment construire et manipuler des tableaux bidimensionnels à travers un exemple d'affichage de formes géométriques.

Copyright © 2016 Eyrolles.

Les tableaux à une dimension

L'étude du chapitre 1, « Stocker une information », montre que pour manipuler plusieurs valeurs à l'intérieur d'un programme, nous devons déclarer autant de variables que de valeurs à traiter. Ainsi, pour stocker les huit notes d'un élève donné, la technique consiste à déclarer huit variables comme suit :

```
double note1, note2, note3, note4, note5, note6, note7, note8;
```

Le fait de déclarer autant de variables qu'il y a de valeurs présente les inconvénients suivants :

- Si le nombre de notes est modifié, il est nécessaire de :
  - Déclarer de nouvelles variables.
  - Placer ces variables dans le programme, afin de les traiter en plus des autres notes.
  - Compiler à nouveau le programme pour que l'interpréteur puisse prendre en compte ces modifications.
- Il faut trouver un nom de variable pour chaque valeur traitée. Imaginez déclarer 1 000 variables portant un nom différent !

Ces inconvénients majeurs sont résolus grâce aux tableaux. En effet, les tableaux sont des structures de données qui regroupent sous un même nom de variable un nombre donné de valeurs de même type. Les tableaux sont proposés par tous les langages de programmation. Ils sont construits par assemblage d'une suite finie de cases mémoire, comme illustré à la figure 9-1.

Chaque case représente l'espace mémoire nécessaire au stockage d'une et d'une seule valeur. Notons que les cases sont réservées en mémoire de façon contiguë.

Déclarer un tableau

Comme toute variable utilisée dans un programme, un tableau doit être déclaré afin de :

- donner un nom à l'ensemble des valeurs à regrouper ;
- définir la taille du tableau de façon à préciser le nombre de valeurs à regrouper ;
- déterminer le type de valeur à mémoriser.

La syntaxe de déclaration d'un tableau est la suivante :

```
TypeDuTableau [] nomDuTableau ;
nomDuTableau = new TypeDuTableau [tailleDuTableau] ;
```

Copyright © 2016 Eyrolles.

En plaçant dans la première instruction les crochets `[]` entre le type et le nom de la variable, nous indiquons au compilateur que la variable `nomDuTableau` représente un tableau. À cette étape, le compilateur réserve un espace mémoire portant le nom du tableau. Cet espace mémoire est susceptible de contenir l'adresse de la première case du tableau.

Ensuite dans la seconde instruction, l'opérateur `new` réserve autant de cases mémoire consécutives qu'il est indiqué entre les `[]` situés en fin d'instruction, soit `tailleDuTableau`. L'opérateur `new` détermine enfin l'adresse de la première case du tableau et la stocke grâce au signe d'affectation, dans la case `nomDuTableau` créée à l'étape précédente.

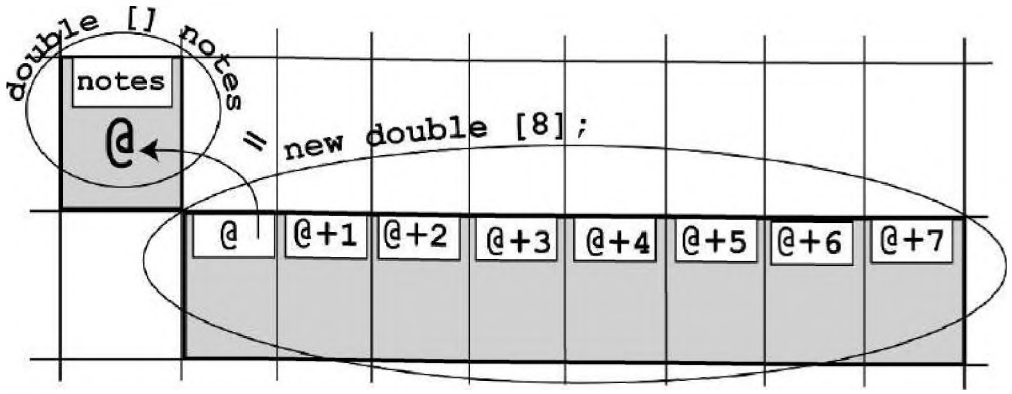


Figure 9-1 *L'opérateur new réserve le nombre de cases mémoire demandé ([8]) et mémorise l'adresse de la première case mémoire dans la variable notes grâce au signe d'affectation.*

Exemple : Déclarer un tableau de huit notes

```
double [] notes ;
notes = new double[8] ;
```

Ces deux instructions réalisent la déclaration d'un tableau ayant pour nom `notes`. Il est composé de 8 cases mémoire pouvant stocker des valeurs de type `double` (voir figure 9-1).

Autres exemples de déclaration

```
// déclarer un tableau de 5 entiers
int [] valeur ;
valeur   =   new int[5];
// déclarer un tableau de 30 réels de simple précision
float [] reel ;
reel     =   new float[30];
// déclarer un tableau de 80 caractères
char [] mot ;
mot      =   new char[80];
```

Copyright © 2016 Eyrolles.

Remarques

- Le nombre de cases réservées correspond au nombre maximal de valeurs à traiter. Lorsque la taille du tableau est fixée après exécution de l'opérateur `new`, il n'est plus possible de la modifier en cours d'exécution du programme.

  Cependant, il est possible de ne pas fixer définitivement la taille du tableau avant compilation en plaçant une variable entre les `[]` au lieu d'une valeur numérique. En effet, il suffit d'écrire :

```
double []   notes;
int   nbNotes ;
Scanner lectureClavier = new Scanner(System.in);
System.out.print("Combien voulez-vous saisir de notes : ");
nbNotes = lectureClavier.nextInt();
notes = new double[nbNotes];
```

  De cette façon, l'utilisateur saisit le nombre de valeurs qu'il souhaite traiter avant la réservation effective des espaces mémoire par l'opérateur `new`. Le programme peut donc voir la taille du tableau varier d'une exécution à l'autre.

- Les tableaux sont des objets. En effet, les tableaux sont définis à l'aide d'une adresse déterminée par l'opérateur `new`. Les tableaux sont donc des objets, au même titre que les `String` et autres objets définis aux chapitres précédents.

  Les objets sont caractérisés par leurs données et les méthodes qui leur sont applicables. Une donnée caractéristique des tableaux est leur taille, c'est-à-dire le nombre de cases. Ainsi, pour connaître la taille d'un tableau, il suffit de placer le terme `.length` derrière le nom du tableau. Par exemple, l'instruction suivante :

```
System.out.print("Nombre de notes = "+ notes.length) ;
```

  affiche à l'écran `Nombre de notes = 8`.

- L'instruction de déclaration :

```
double [] notes = new double[8];
```

  est équivalente à la suite d'instructions :

```
double [] notes ;
notes = new double[8] ;
```

Manipuler un tableau

Un tableau est un ensemble de cases mémoire. Chaque case constituant un élément du tableau est identique à une variable. Il est possible de manipuler chaque case du tableau de façon à :

- placer une valeur dans une case du tableau à l'aide de l'affectation ;

Copyright © 2016 Eyrolles.

- utiliser un élément du tableau dans le calcul d'une expression mathématique ;
- afficher un élément du tableau.

Accéder aux éléments d'un tableau

Sachant que `nomDuTableau[0]` représente la première case du tableau, l'accès à la *nième* case s'écrit `nomDuTableau[n]`.

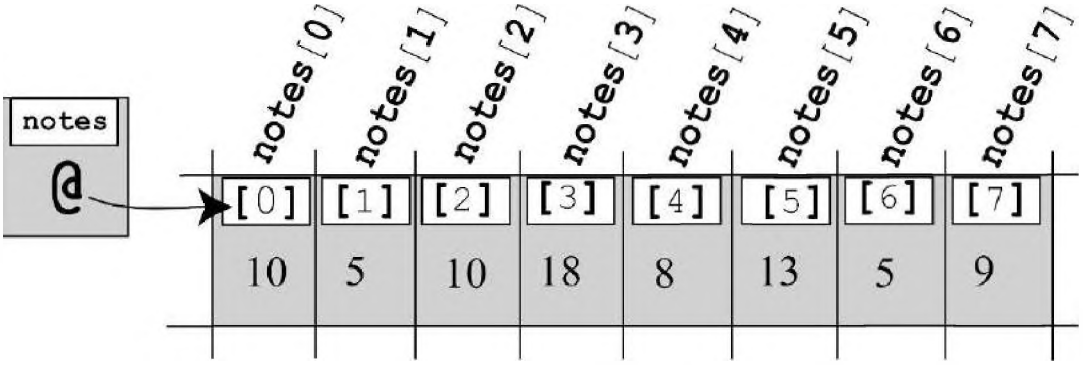


Figure 9-2 *note est le nom du tableau, et les notes 10, 5, ..., 9 sont des valeurs placées à l'aide du signe d'affectation dans les cases numérotées respectivement 0, 1, ..., 7 (indices).*

Par exemple, l'instruction :

```
note[0] = 10 ;
```

mémorise la première note d'un étudiant dans la première case du tableau (`notes[0]`). De la même façon, la deuxième note est stockée grâce à l'affectation :

```
note[1] = 5 ;
```

Et ainsi de suite, jusqu'à stocker la huitième et dernière note à l'aide de l'instruction :

```
note[7] = 9 ;
```

Les valeurs placées entre les crochets `[]` sont appelées les **indices** du tableau.

Remarque La première case du tableau est numérotée à partir de `0` et non de `1` (voir figure 9-2). L'indice du tableau varie donc entre `0` et `length-1`.

Les éléments d'un tableau étant ordonnés grâce aux indices, il est possible d'y accéder à l'aide de constructions itératives (boucle `for`), comme le montre l'exemple suivant.

Copyright © 2016 Eyrolles.

Exemple : extrait d'un programme

```
Scanner lectureClavier = new Scanner(System.in);
System.out.print("Combien de notes voulez-vous saisir : ");
int nombre = lectureClavier.nextInt();
notes = new double [nombre];
for (int i = 0; i < notes.length; i++) {
  System.out.print("Entrer la note  n° "+ (i+1)+ " :  ");
  notes[i] = lectureClavier.nextDouble();
}
```

Exemple : résultat de l'exécution

Les caractères grisés sont des valeurs choisies par l'utilisateur.

```
Combien de notes voulez-vous saisir : 4
Entrer la note  n° 1 : 14
Entrer la note  n° 2 : 10
Entrer la note  n° 3 : 12
Entrer la note  n° 4 : 8
```

Une fois le nombre de notes déterminé grâce aux deux premières instructions, le programme entre dans une boucle `for`. La variable `i` correspond au compteur de boucle. Elle varie entre `0` et `notes.length-1` (soit 3), puisque la condition de continuation précise que `i` doit être strictement inférieure à `notes.length` (soit 4).

À chaque tour de boucle, la variable `i` prend la valeur de l'indice du tableau (`notes[i]`). Les valeurs saisies au clavier sont alors placées une à une dans chaque case du tableau.

Parce qu'il n'est pas courant de compter des valeurs à partir de `0`, l'affichage demandant d'entrer une note débute à `1`, et non à `0`, grâce à l'expression `(i+1)` placée dans la méthode `System.out.print()`. Il ne s'agit là que d'un artifice de présentation, la première note étant stockée en réalité en `notes[0]`.

Remarque

L'utilisation de la donnée `length` permet d'éviter tout problème de dépassement de taille. En effet, si un tableau est composé de quatre cases, il n'est pas possible de placer une valeur à l'indice 4 ou 5. Le fait d'écrire `notes[4]` génère une erreur d'exécution du type : `java.lang.ArrayIndexOutOfBoundsException`, qui montre que l'interpréteur Java a détecté que l'indice du tableau était en dehors des limites définies au moment de sa création.

Copyright © 2016 Eyrolles.

Initialiser un tableau

Lors de la déclaration d'un tableau, il est possible de l'initialiser directement de la façon suivante :

```
double [] notes = {10, 12.5, 5, 8.5, 16, 0, 13, 7} ;
```

Les cases mémoire sont réservées et initialisées, dans l'ordre, à l'aide des valeurs placées entre les `{}` et séparées par des virgules. De cette façon, le tableau `notes` contient les valeurs suivantes :

```
notes[0] vaut   10    notes[4]   vaut   16
notes[1] vaut   12.5  notes[5]   vaut   0
notes[2] vaut   5     notes[6]   vaut   13
notes[3] vaut   8.5   notes[7]   vaut   7
```

Signalons que la donnée `notes.length` prend automatiquement la valeur `8`.

La nouvelle boucle for

Avec la version 1.5 du compilateur (jdk1.5.0), le langage Java propose une nouvelle syntaxe pour la boucle `for`, qui simplifie le code d'écriture des parcours des tableaux.

Ainsi le parcours du tableau notes initialisé précédemment, s'effectue en utilisant la syntaxe suivante :

```
for (int valeur : notes){
 System.out.println(valeur);
}
```

La boucle `for` ainsi écrite n'utilise plus de compteur de boucle ni de test de fin de boucle. Ceux-ci sont gérés de façon transparente pour le programmeur.

Le tableau `notes` est parcouru élément par élément, du premier jusqu'au dernier élément. La valeur de chacun des éléments est enregistrée tour à tour dans la variable valeur qui est ensuite affichée. Plus simplement, vous pouvez traduire cette nouvelle boucle for en utilisant la formule suivante :

« Pour chaque valeur du tableau notes, afficher la valeur ».

Remarque

Si l'on souhaite n'examiner qu'une partie du tableau comme par exemple un élément sur deux ou encore la première moitié du tableau, nous devons connaître la valeur du compteur de boucles, il convient alors d'utiliser une boucle `for` classique. Dans la suite de cet ouvrage, nous utilisons la nouvelle forme de la boucle `for` chaque fois que cela est possible.

Copyright © 2016 Eyrolles.

Les tableaux et les opérations arithmétiques

La somme, la soustraction, la division ou la multiplication directes de deux tableaux sont des opérations impossibles. En effet, chaque opération doit être réalisée élément par élément, comme le montre le tableau suivant :

Correcte	Impossible

`int [] tab1 = new int[10] ;`
`int [] tab2 = new int[10];`
`int [] somme = new int[10];`
`for (i = 0 ; i < 10 ; i++)`
`somme[i] = tab1[i] + tab2[i];`
	`int [] tab1 = new int[10] ;`
	`int [] tab2 = new int[10];`
	`int [] somme = new int[10];`
	~~`somme = tab1 + tab2;`~~

Quelques techniques utiles

Le stockage et l'utilisation des données à travers la structure des tableaux offrent de nombreux avantages. Elles requièrent aussi certaines techniques de manipulation, qui sont développées ci-après.

La ligne de commande

Au cours du chapitre « Naissance d'un programme », vous avez dû admettre un certain nombre de termes du langage Java et, en particulier, la syntaxe de l'instruction suivante :

```
public static void main(String [] argument)
```

Cette instruction correspond à la définition de l'en-tête de la fonction `main()`. Vous êtes en mesure maintenant de déchiffrer chacun de ses termes pour en comprendre l'utilité :

- Le mot-clé `public` précise au compilateur que la fonction `main()` est accessible depuis l'extérieur de la classe où elle est définie. En particulier, l'interpréteur Java peut y accéder pour l'exécuter.
- Le terme `static` explique que la fonction `main()` ne peut pas être copiée plusieurs fois en mémoire. Elle ne peut pas être associée à un objet ni être instanciée, c'est-à-dire qu'il n'est pas possible d'écrire `unObjet.main()`.
- La fonction `main()` ne fournit pas de résultat, et c'est pourquoi elle est définie comme `void`.
- Pour finir, elle possède, entre `()`, un paramètre défini comme tableau de type `String`. Ce paramètre est utilisé pour passer des données en **ligne de commande** lors du lancement de la commande d'exécution du programme.

Copyright © 2016 Eyrolles.

Qu'est-ce qu'une ligne de commande ?

Une ligne de commande est écrite au clavier sous la forme d'une instruction précise. C'est un ordre transmis à l'ordinateur par l'utilisateur. Sous Unix, les commandes sont très utilisées. Elles le sont beaucoup moins sous Windows et sont inexistantes sous Mac OS. C'est pourquoi les utilisateurs de stations de travail Unix n'ont aucune difficulté à comprendre ce qu'est une commande, ce qui n'est pas le cas des utilisateurs de PC (sous Windows) ou de Macintosh.

Aujourd'hui grâce aux écrans graphiques, l'utilisateur communique facilement avec l'ordinateur. Pour savoir ce que contient un dossier, il lui suffit d'ouvrir la fenêtre associée à ce dossier. Les ordres passés à l'ordinateur sont essentiellement des ordres générés par la souris au travers de fenêtres graphiques.

Les lignes de commande sont équivalentes, bien que moins conviviales, à cette communication graphique. Elles permettent surtout d'obtenir des résultats plus précis. Ainsi, les commandes :

- `ls *.java`, dans une fenêtre de commandes Unix ;
- `dir *.java`, dans une fenêtre « commandes MS-DOS » ;

ont pour résultat d'afficher tous les noms de fichiers finissant par `.java` contenus dans le répertoire courant.

Plus précisément, notons qu'une commande s'écrit toujours de la façon suivante :

```
nomDeLaCommande paramètresEventuels
```

Le nom d'une commande correspond au nom du programme qui réalise l'action souhaitée. Les paramètres sont utilisés pour affiner son résultat. Dans notre exemple, `*.java` est un paramètre des commandes `ls` ou `dir`, qui permet d'expliquer à l'ordinateur que vous souhaitez voir s'afficher uniquement les noms de fichiers finissant par `.java`.

Passer des paramètres à un programme Java

De la même façon, comme expliqué à la section « Exécuter un programme » du chapitre introductif « Naissance d'un programme », l'exécution d'un programme Java en dehors d'un environnement de travail passe aussi par une commande dont la syntaxe est :

```
java nomdel'application
```

L'interpréteur Java autorise aussi la commande :

```
java nomdel'application p0 p1 p2… pN
```

Dans ce cas, les valeurs `p0`, `p1`, `p2`, …, `pN`, toutes séparées par des espaces, sont considérées comme paramètres de la commande `java nomdel'application`. Ces derniers sont transmis à la fonction `main()` par l'intermédiaire du tableau de `String` défini en paramètre de la fonction.

Copyright © 2016 Eyrolles.

Si l'en-tête a pour forme :

```
public static void main(String [] argument)
```

le paramètre `p0` est stocké en `argument[0]`, `p1` en `argument[1]`, ... et `pN` en `argument[N]`. Les valeurs ainsi passées sont mémorisées sous forme de chaînes de caractères.

Exemple : une commande qui calcule

Pour mieux comprendre cette transmission de valeurs, reprenons le corrigé de l'exercice 3 du chapitre 6, « Fonctions, notions avancées », qui simule une calculette. Transformons ce programme de sorte qu'il puisse effectuer l'opération à partir de valeurs passées en paramètres lors de la commande d'exécution du programme. Supposons que cette commande s'écrive :

```
java Calculette 1 + 2
```

L'ordre des paramètres ainsi passés est important. En effet, nous devons traiter les paramètres de la fonction `main()` de la façon suivante :

- Les premier et troisième paramètres doivent être interprétés comme étant les valeurs numériques de l'opération à calculer.
- Le deuxième paramètre doit correspondre à l'opérateur.

Sachant cela, le programme s'écrit de la façon suivante.

Exemple : le code source

```
import java.util.*;
public class Calculette {
 public static void main(String [] argument) {
  int a, b;
  char opérateur;
  double calcul;
  Scanner lectureClavier = new Scanner(System.in);
   if (argument.length > 0) {
    a = Integer.parseInt(argument[0]);
    opérateur = argument[1].charAt(0);
    b = Integer.parseInt(argument[2]);

  else {
    opérateur = menu();
    System.out.println("Entrer la premiere valeur ");
    a = lectureClavier.nextInt();
    System.out.println("Entrer la seconde valeur ");
    b = lectureClavier.nextInt();
  }
```

Copyright © 2016 Eyrolles.

```
    calcul = calculer(a, b, opérateur );
    afficher(a, b, opérateur, calcul);
  }

  public static double calculer (int x, int y, char o) {
// voir corrigé de l'exercice 3 du chapitre "Fonctions, notions
//avancées"
  }

  public static void afficher(int x, int y, char o, double r) {
// voir corrigé de l'exercice 3 du chapitre "Fonctions, notions
//avancées"
  }
  public static char menu() {
// voir corrigé de l'exercice 3 du chapitre "Fonctions, notions
//avancées"
  }
}
```

Pour traiter les paramètres passés en ligne de commande, il est nécessaire de détecter si des paramètres ont été effectivement passés. Pour ce faire, l'idée est de regarder la taille du tableau `argument`, de la façon suivante :

- Si celle-ci n'est pas nulle (supérieure strictement à 0), cela signifie que le tableau contient des paramètres passés en ligne de commande. Dans ce cas, nous traitons chacun des arguments de sorte que le calcul puisse être effectué.

  Les éléments `argument[0]` et `argument[2]` contiennent par hypothèse les deux valeurs numériques. Or, celles-ci sont stockées sous forme de suites de caractères, le tableau `argument` étant de type `String`. Les valeurs doivent donc être « traduites » en format numérique. Comme nous souhaitons obtenir des valeurs entières, la méthode proposée par le langage Java a pour nom `Integer.parseInt()`. Ainsi, les instructions :

  ```
  a = Integer.parseInt(argument[0]);
  b = Integer.parseInt(argument[2]);
  ```

  permettent la traduction de la suite de caractères contenue dans `argument[0]` et `argument[2]` en valeurs numériques et de placer ces valeurs dans les variables `a` et `b` déclarées de type `int`. Compte tenu des paramètres passés en ligne de commande, les variables `a` et `b` ont donc pour valeurs respectives `1` et `2`.

  Le caractère correspondant à l'opérateur est stocké dans `argument[1]`. Nous devons le transformer en `char` puisqu'un opérateur est formé d'un seul caractère. Cette transformation

Copyright © 2016 Eyrolles.

est réalisée par une méthode de la classe `String`, appelée `charAt()`, qui retourne le caractère placé à la position spécifiée en paramètre. Ainsi, l'instruction :

```
opérateur = argument[1].charAt(0);
```

place dans la variable `opérateur` le premier caractère du mot stocké dans `argument[1]`, soit, pour notre exemple, le caractère +.

- Si la taille du tableau `argument` est nulle, cela signifie qu'aucun paramètre n'a été transmis. Le bloc `else` est exécuté, et les valeurs sont saisies au clavier, comme cela était le cas en fin de correction de l'exercice 3 du chapitre 6.

Pour finir, lorsque les valeurs choisies sont placées dans les variables `a`, `b` et `opérateur`, à l'aide des paramètres ou du clavier, le calcul de l'opération est réalisé et le résultat est affiché. Dans l'exemple, vous obtenez :

```
java Calculette 1 + 2
1 + 2 = 3
```

Précisons en outre que cette commande doit être obligatoirement lancée dans le répertoire où se trouve le fichier `Calculette.class`.

Question Que se passe-t-il si l'on tape la commande :

```
java Calculette 1 + 2 - 3
```

Réponse Le programme s'exécute comme précédemment. Il affiche le résultat de la façon suivante :

```
java Calculette 1 + 2
1 + 2 = 3
```

La soustraction –3 n'est pas prise en compte, puisque seuls `argument[0]`, `argument[1]` et `argument[2]` sont traités et utilisés dans le code source du programme.

Question Que se passe-t-il si l'on tape la commande :

```
java Calculette 1
```

Réponse L'interpréteur affiche le message d'erreur suivant : `Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException at Calculette.main`

En effet, le programme est appelé avec une ligne de commande qui n'utilise qu'un seul paramètre. De ce fait, seul `argument[0]` est créé en mémoire. Lorsqu' ensuite le programme traite `argument[1]` puis `argument[2]`, l'interpréteur indique que les indices *1* et *2* n'existent pas. Le tableau *argument* n'ayant qu'une seule case, il ne peut accéder à d'autres cases sans dépasser les limites de la taille du tableau.

Copyright © 2016 Eyrolles.

Trier un ensemble de données

L'atout principal de l'ordinateur est sa faculté à traiter un très grand nombre de données en des temps très rapides. Ces traitements sont, par exemple, la recherche d'éléments dans un ensemble en suivant des contraintes choisies par l'utilisateur ou encore le tri d'éléments en fonction d'un critère déterminé.

Pour comprendre le fonctionnement interne de ces traitements, nous étudions ici l'algorithme du « tri par extraction simple », qui utilise les techniques de recherche d'un élément dans un ensemble de données, d'échange de valeurs et de tri.

Cahier des charges

L'objectif du programme est de réaliser le classement par moyenne d'une classe d'étudiants. Pour cela, nous devons tout d'abord définir ce qu'est un étudiant (voir la section « La classe Etudiant ») pour décrire ensuite une classe d'étudiants (voir la section « La classe Cursus »). Cela fait, il devient possible de trier une classe d'étudiants selon leur moyenne (voir la section « La méthode du tri par extraction simple »).

La classe Etudiant

Un étudiant est défini par son nom (`String`), son prénom (`String`), un ensemble de notes (un tableau de `double`) et une moyenne (`double`). Ces caractéristiques constituent l'ensemble des données du type `Etudiant`.

Les comportements d'un étudiant permettent l'initialisation et l'affichage de ses caractéristiques, ainsi que le calcul de sa moyenne.

Par conséquent, nous décrivons comme suit la classe `Etudiant` :

```
import java.util.*;
public class Etudiant    {
 // Les données caractéristiques
 private String nom, prénom;
 private double [] notes, moyenne;

 // Les comportements
 public Etudiant()       {
  Scanner lectureClavier = new Scanner(System.in);
  System.out.print("Entrer le nom de l'etudiant : ");
  nom = lectureClavier.next();
  System.out.print("Entrer le prénom de l'etudiant : ");
  prénom = lectureClavier.next();
  System.out.print("Combien de notes pour l'etudiant ");
```

Copyright © 2016 Eyrolles.

```
    System.out.print(prénom + " " + nom + " :  ");
    int nombre = lectureClavier.nextInt();
    notes = new double [nombre];
    for (int i = 0; i < notes.length; i ++){
     System.out.print("Entrer la note  n° "+ (i + 1) + " : ");
     notes[i] = lectureClavier.nextDouble();
    }
    moyenne = calculMoyenne();
   }
   private double calculMoyenne() {
    double somme = 0.0;
    for(double valeurNote : notes) somme = somme + valeurNote;
    return somme/notes.length;
   }
   public void afficheUnEtudiant() {
    System.out.print("Les notes  de " + prénom + " " + nom
     ,                                 + " sont : ");
    for(double valeurNote : notes)
      System.out.print(" " + valeurNote);
    System.out.println();
    System.out.println("Sa moyenne vaut " + moyenne);
   }
  public double quelleMoyenne() {
          return moyenne;
   }
  } // Fin de class Etudiant
```

La classe `Etudiant` définit les quatre méthodes suivantes :

- `Etudiant()`. C'est le constructeur de la classe, qui permet d'initialiser l'ensemble des données de la classe `Etudiant` en demandant la saisie au clavier des nom et prénom de l'étudiant, ainsi que de l'ensemble de ses notes. Le nombre de notes peut varier d'un étudiant à un autre, puisque la valeur `nombre` est saisie en cours d'exécution.
- `calculMoyenne()`. Une fois les données saisies, le programme calcule la moyenne à l'intérieur du constructeur, grâce à la méthode `calculMoyenne()`. Cette méthode est déclarée en `private` car, pour des raisons de sécurité, ce calcul ne peut être réalisé qu'à l'intérieur de la classe `Etudiant`.
- `afficheUnEtudiant()`. Affiche à l'écran les caractéristiques d'un étudiant.
- `quelleMoyenne()`. La donnée `moyenne` étant protégée (`private`), la méthode `quelleMoyenne()` permet l'accès en consultation, depuis l'extérieur de la classe, de la valeur mémorisée.

Copyright © 2016 Eyrolles.

La classe Cursus

Une classe d'étudiants est définie par un ensemble d'étudiants, c'est-à-dire un tableau d'objets `Etudiant`.

Les comportements de la classe `Cursus` permettent l'initialisation, l'affichage de ses données, ainsi que le classement des étudiants dans l'ordre croissant des moyennes.

La classe `Cursus` est décrite comme suit :

```
import java.util.*;
public class Cursus {
 private Etudiant [] liste;
 public Classe()   {
  Scanner lectureClavier = new Scanner(System.in);
  System.out.print("Nombre d'etudiants : ");
  int nbetudiants = lectureClavier.nextInt();
  liste = new Etudiant[nbetudiants];
  for(int i = 0; i < liste.length; i++) liste[i] = new Etudiant();
 }
 public void afficheLesEtudiants() {
  for (Etudiant e : liste) e.afficheUnEtudiant();
 }
} // Fin de class Cursus
```

La donnée `liste` de la classe `Cursus` est un tableau d'objets de type `Etudiant`. Il s'agit donc là d'un tableau particulier, puisque chaque case du tableau ne correspond pas à une valeur numérique simple mais à l'ensemble des données caractéristiques d'un étudiant.

En réalité, chaque case du tableau `liste` contient l'adresse d'un objet de type `Etudiant`, comme illustré à la figure 9-3. Cette opération est effectuée par le constructeur `Classe()`.

Ce dernier réalise la création du tableau en deux étapes. Ainsi, l'instruction :

```
liste = new Etudiant[nbetudiants];
```

crée une case mémoire `liste`, qui contient l'adresse de la première case mémoire du tableau. Ce tableau est de type `Etudiant`. Il est donc destiné à stocker les adresses des objets de type `Etudiant`.

Ensuite, la boucle :

```
for(int i = 0; i < liste.length; i++) liste[i] = new Etudiant();
```

réalise, en faisant appel au constructeur de la classe `Etudiant`, la création en mémoire des objets de type `Etudiant`, ainsi que la saisie des informations relatives à chaque étudiant.

Copyright © 2016 Eyrolles.

Pour finir, chaque adresse produite par l'opérateur new dans la boucle `for` est placée dans chacune des cases mémoire du tableau `liste` (`liste[i]`).

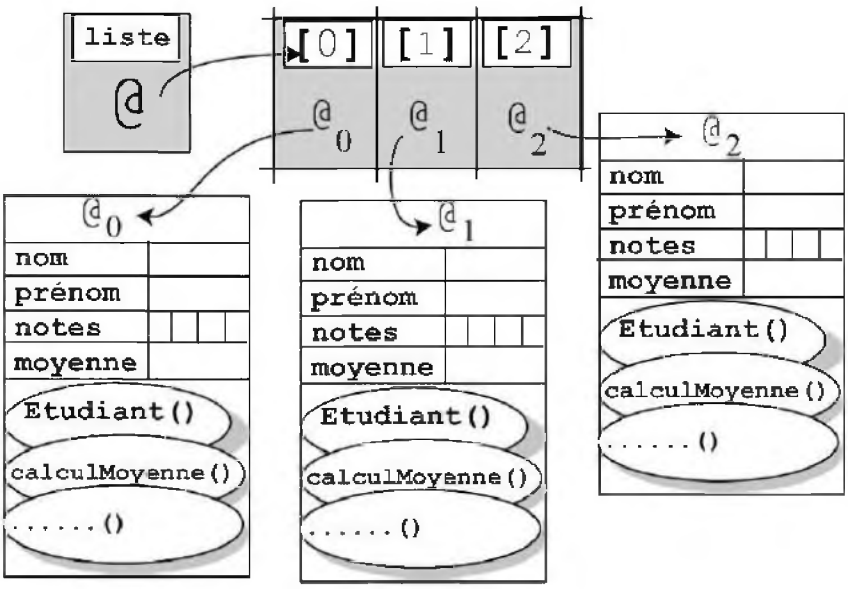


Figure 9-3 *Le tableau liste est un tableau d'objets. Chaque case du tableau mémorise l'adresse d'un objet Etudiant.*

La méthode `afficheLesEtudiants()` permet l'affichage des informations relatives aux étudiants en faisant appel à la méthode `afficheUnEtudiant()` de la classe `Etudiant`. Cette méthode affiche les caractéristiques d'un étudiant.

Grâce à la nouvelle syntaxe de la boucle `for`, il n'est plus nécessaire d'utiliser de compteur, ni de test de fin de boucle. Le tableau d'étudiants `liste` est parcouru élément par élément. Chaque élément est enregistré tour à tour dans un objet `e` de type `Etudiant` sur lequel la méthode `afficheUnEtudiant()` est appliquée. Ainsi chaque étudiant de la liste est affiché.

La méthode du tri par extraction simple

Grâce aux classes `Etudiant` et `Cursus`, nous sommes en mesure de créer un ensemble d'étudiants possédant chacun un nombre de notes et une moyenne. Notre objectif étant d'afficher un classement des étudiants par ordre croissant des moyennes, examinons comment trier l'ensemble des moyennes d'une classe.

L'algorithme du tri par extraction simple se décrit de la façon suivante (voir figure 9-4) :

- Parcourir l'ensemble des moyennes de la classe afin de trouver la plus petite.

Copyright © 2016 Eyrolles.

- Une fois trouvée, échanger cette valeur avec celle placée au tout début du tableau, de façon à être sûr que la moyenne la plus faible se trouve en début du tableau.
- Recommencer ce même traitement sur l'ensemble des moyennes moins la première, puisqu'elle vient d'être traitée.

Deux étapes sont nécessaires pour traduire cet algorithme en langage Java. Elles sont décrites ci-après, aux sections « Recherche du plus petit élément dans une liste » et « Échange de la plus petite valeur avec un élément de la liste ».

Recherche du plus petit élément dans une liste

Pour trouver la plus petite valeur d'un ensemble de valeurs, il suffit de comparer chaque valeur de la liste avec celle tout d'abord située en début de liste puis avec une plus petite, si elle existe, dans la liste. C'est ce que réalise la boucle suivante :

```
int indiceDuMin = 0 ;
for(int j = 1; j < liste.length; j++)
    if (liste[j].quelleMoyenne() < liste[indiceDuMin].quelle
        Moyenne())indiceDuMin = j;
```

Ainsi, chaque valeur du tableau (`j` variant de `1` à `liste.length`) est comparée avec la première valeur du tableau (`indiceDuMin` valant `0` en début de boucle). Si la comparaison montre que la valeur placée à l'indice `j` est plus petite que celle placée en `indiceDuMin`, alors l'indice de cette plus petite valeur est stockée dans la variable `indiceDuMin`. Le test suivant compare la valeur suivante avec la plus petite valeur qui vient d'être détectée.

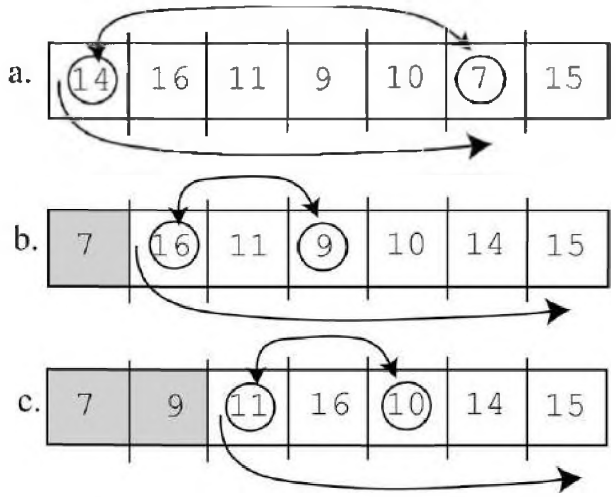


Figure 9-4 *a. Parcours du tableau entier afin de déterminer la plus petite valeur puis échange de cette dernière avec la valeur stockée en première position dans le tableau. b. Même traitement à partir de la deuxième case du tableau. c. Même traitement à partir de la troisième case du tableau.*

Copyright © 2016 Eyrolles.

Grâce à cette boucle, la recherche de la plus petite valeur est réalisée sur l'intégralité du tableau. Or, dans l'algorithme du tri présenté ci-dessus, cette recherche doit être réalisée dans un premier temps sur l'intégralité de la liste, puis à partir du deuxième élément, ensuite à partir du troisième élément, etc.

Cette boucle de recherche doit être placée à l'intérieur d'une méthode, de façon à pouvoir être exécutée plusieurs fois, chacune des exécutions variant en fonction d'un paramètre qui précise l'indice où débute la recherche. La méthode `ouEstLePlusPetit()`, présentée ci-dessous et à insérer dans la classe `Cursus`, réalise cette recherche.

```
private int ouEstLePlusPetit(int debut)  {
  int indiceDuMin = debut, j;
  for(j = debut+1; j < liste.length; j++)
   if (liste[j].quelleMoyenne() < liste[indiceDuMin].quelleMoyenne())
      indiceDuMin = j;
  return indiceDuMin;
 }
```

Lorsque le programme sort de la boucle `for`, `indiceDuMin` représente l'indice de la plus petite moyenne dans le tableau `liste`. Cette valeur est alors retournée à la fonction appelante, qui l'utilise pour réaliser l'échange des valeurs.

Notez que la méthode `ouEstLePlusPetit()` est déclarée en mode `private`. Cette méthode n'est pas un comportement caractéristique d'une classe d'étudiants mais un traitement interne destiné à obtenir un classement de l'ensemble des étudiants.

Échange de la plus petite valeur avec un élément de la liste

Connaissant l'indice où se trouve la plus petite moyenne, nous devons échanger cette valeur avec celle correspondant au début de la recherche. Ce traitement est réalisé sur l'ensemble des étudiants en faisant varier l'indice du début de recherche de la première valeur du tableau jusqu'à la dernière. La méthode `classerParMoyenne()`, qui s'insère dans la classe `Cursus`, réalise ces opérations :

```
public void classerParMoyenne()  {
  int indiceDuPlusPetit ;
  Etudiant tmp;
  for(int i = 0; i < liste.length; i ++)   {
     indiceDuPlusPetit = ouEstLePlusPetit(i);
     tmp = liste[i];
     liste[i] = liste[indiceDuPlusPetit];
     liste[indiceDuPlusPetit] = tmp;
  }
 }
```

Grâce à la boucle `for`, le programme parcourt l'ensemble des étudiants de la classe. Ainsi, pour chaque étudiant de la `liste`, la boucle réalise :

Copyright © 2016 Eyrolles.

- la recherche de la plus petite moyenne (`ouEstLePlusPetit()`) à partir de l'indice `i`, correspondant à l'indice de début de recherche ;
- l'échange dans la liste des données concernant l'étudiant ayant la plus petite moyenne (`indiceDuPlusPetit`) avec les données de l'étudiant placé à l'indice `i` (indice du début de recherche).

Sans revenir sur le mécanisme d'échange des données, observez que grâce au regroupement des données sous forme d'objets, les opérations réalisent non seulement l'échange des moyennes des étudiants, mais aussi l'ensemble des données décrivant chaque étudiant, c'est-à-dire ses nom, prénom et notes. En effet, ce sont ici les adresses de chaque objet « `Etudiant` » qui sont échangées, et non pas simplement les moyennes.

Pour en savoir plus Sur l'échange de deux valeurs, voir au chapitre 1, « Stocker une information », la section « Échanger les valeurs de deux variables ».

L'application GestionCursus

Afin de vérifier le bon fonctionnement des classes `Etudiant` et `Cursus`, il est nécessaire de construire une application qui utilise des instances de ces classes. Examinons la classe `GestionCursus`, composée d'une fonction `main()`, dans laquelle est déclaré un objet `C` de type `Cursus`.

```
public class GestionCursus  {
 public static void main(String [] argument)  {
  Cursus C = new Cursus();
  System.out.println("-------------  Recapitulatif  -----------");
  C.afficheLesEtudiants();
  C.classerParMoyenne();
  System.out.println("---------------  Classement  ------------");
  C.afficheLesEtudiants();
 }
} // Fin de class GestionClasse
```

En appelant le constructeur `Cursus()`, le programme demande la saisie du nombre d'étudiants. Puis, pour chaque étudiant, il fait appel au constructeur `Etudiant()`, qui demande la saisie des nom, prénom et notes de l'étudiant concerné.

À la sortie du constructeur `Cursus()`, le programme est en mesure d'afficher, grâce à l'instruction `C.afficheLesEtudiants()`, toutes les informations relatives à chaque étudiant de la classe `C`.

Ensuite, les étudiants sont classés par ordre croissant de moyenne grâce à l'appel de la méthode `classerParMoyenne()`, appliquée à la classe `C`. L'affichage de la liste des étudiants permet ensuite de vérifier que le tri a été correctement réalisé.

Copyright © 2016 Eyrolles.

Exécution de l'application : résultats

```
Nombre d'etudiants : 4
Entrer le nom de l'etudiant : M.
Entrer le prenom de l'etudiant : Olivier
Combien de notes pour l'etudiant Olivier M. : 2
Entrer la note  n° 1 : 13
Entrer la note  n° 2 : 15
Entrer le nom de l'etudiant : B.
Entrer le prenom de l'etudiant : Arnaud
Combien de notes pour l'etudiant Arnaud B. : 2
Entrer la note  n° 1 : 16
Entrer la note  n° 2 : 18
Entrer le nom de l'etudiant : R.
Entrer le prenom de l'etudiant : Mathieu
Combien de notes pour l'etudiant Mathieu R. : 2
Entrer la note  n° 1 : 16
Entrer la note  n° 2 : 16
Entrer le nom de l'etudiant : D.
Entrer le prenom de l'etudiant : Jocelyn
Combien de notes pour l'etudiant Jocelyn D. : 2
Entrer la note  n° 1 : 10
Entrer la note  n° 2 : 12
--------------  Recapitulatif  --------------
Les notes de Olivier M. sont : 13.0 15.0
Sa moyenne vaut 14.0
Les notes de Arnaud B. sont : 16.0 18.0
Sa moyenne vaut 17.0
Les notes de Mathieu R. sont : 16.0 16.0
Sa moyenne vaut 16.0
Les notes de Jocelyn D. sont : 10.0 12.0
Sa moyenne vaut 11.
---------------  Classement  ----------------
Les notes de Jocelyn D. sont : 10.0 12.0
```

Copyright © 2016 Eyrolles.

```
Sa moyenne vaut 11.0
Les notes de Olivier M. sont : 13.0 15.0
Sa moyenne vaut 14.0
Les notes de Mathieu R. sont : 16.0 16.0
Sa moyenne vaut 16.0
Les notes d'Elena P. sont : 16.0 18.0
Sa moyenne vaut 17.0
```

Les tableaux à deux dimensions

Vous venez de voir les tableaux à une seule dimension, représentés comme une liste horizontale ou verticale d'éléments de même type. Il est possible avec Java, de travailler aussi avec des tableaux de deux, trois, voire *n* dimensions. Pour simplifier, nous allons étudier les tableaux à deux dimensions.

Par définition, un tableau à deux dimensions s'organise non plus sur une seule ligne mais sur des lignes et des colonnes. Le croisement d'une ligne et d'une colonne détermine un élément donné du tableau.

Déclaration d'un tableau à deux dimensions

Pour déclarer un tableau à deux dimensions, la syntaxe est la suivante :

```
int [][] donnée = new int [3][5];
```

La syntaxe est pratiquement identique à la déclaration d'un tableau à une dimension. La seule différence consiste en l'ajout de `[]` supplémentaires pour signifier au compilateur que le tableau est à de deux dimensions.

Les valeurs numériques placées entre `[]` derrière l'opérateur `new` indiquent respectivement, le nombre de lignes puis de colonnes.

L'instruction de déclaration décrite ci-dessus réserve en mémoire un tableau nommé `donnée`, composé de 3 lignes et de 5 colonnes. Chaque élément du tableau étant un entier, l'opérateur `new` réserve 3 * 5, soit 15 cases mémoire de la taille d'un entier.

Remarque

Le nombre de lignes d'un tableau est donné par l'expression `donnée.length`, alors que le nombre de colonnes est déterminé par l'expression `donnée[0].length`. En effet, le nombre de colonnes d'un tableau correspond au nombre d'éléments placés sur une ligne (voir figure 9-5).

Copyright © 2016 Eyrolles.

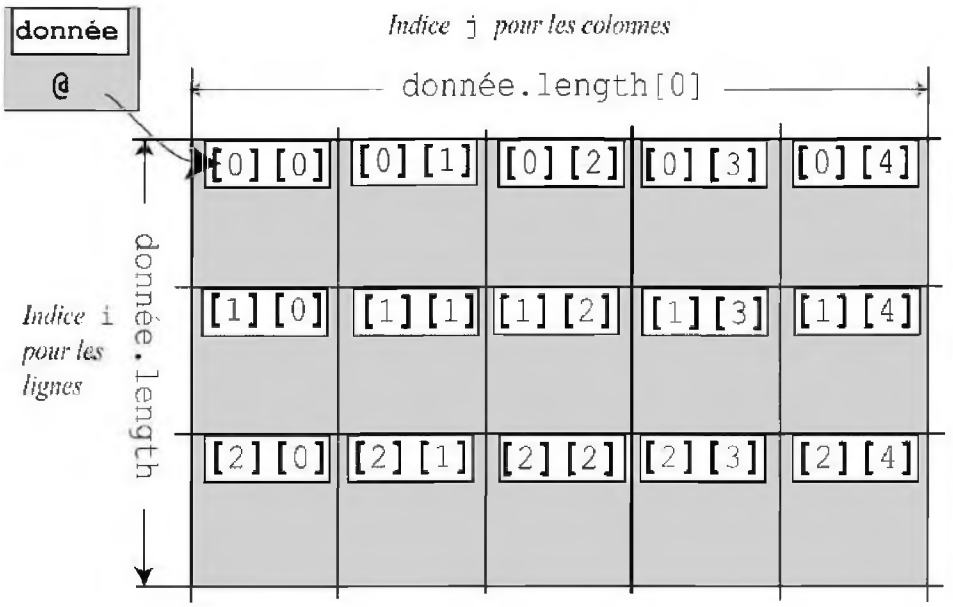


Figure 9-5 *Un tableau s'organise sur des lignes et des colonnes numérotées à partir de [0][0].*

Accéder aux éléments d'un tableau

Pour initialiser, modifier ou consulter la valeur d'un élément d'un tableau, il convient d'utiliser deux indices : un indice pour les lignes et un indice pour les colonnes. Chaque indice étant contrôlé par une boucle `for`, la technique consiste à imbriquer deux boucles de la façon suivante :

```
for (int i = 0; i < donnée.length; i++)
  for (int j = 0; j < donnée[0].length; j++)
   // donnée[i][j] = uneValeur ;
```

La boucle `j` est imbriquée dans la boucle `i`. Les variables `i` et `j` sont les compteurs de boucles qui contrôlent respectivement les lignes et colonnes du tableau `donnée`.

Pour mieux comprendre les mécanismes de manipulation des tableaux et en particulier, le déroulement des valeurs des indices à l'intérieur des boucles `for`, examinons l'exemple suivant.

Exemple : dessiner un sapin

Les tableaux à deux dimensions sont très souvent utilisés pour stocker les images. En effet, une image affichée à l'écran correspond en réalité à une surface découpée en lignes et

Copyright © 2016 Eyrolles.

colonnes. La donnée numérique se situant à la croisée de ces lignes et colonnes représente un point de l'image et a pour valeur la couleur d'affichage à l'écran.

Cahier des charges

L'objectif de cet exemple est de dessiner à l'écran un sapin de Noël décoré, comme l'illustre la figure suivante :

```
%
...
.....
.%.%...
........%
%.%.....%.
```

Pour simplifier à l'extrême la lisibilité du programme, nous n'utilisons que de simples caractères alphanumériques pour afficher notre sapin. C'est pourquoi son affichage reste assez sommaire.

Pour en savoir plus Et pour voir des sapins plus élaborés, reportez-vous au chapitre 11, « Dessiner des objets ».

Créer et afficher un triangle composé de trois lignes

Fidèles au principe de décomposition d'un problème, nous allons chercher dans un premier temps à afficher la forme suivante :

```
.
...
.....
```

L'affichage de cette forme correspond à un triangle. Sa structure interne est définie en mémoire à l'aide d'un tableau à deux dimensions. Il s'agit d'un tableau composé de 3 lignes et de 5 colonnes, comme l'illustre le tableau suivant :

```
00100
01110
11111
```

Ce tableau est constitué de valeurs numériques placées de telle façon que le programme dessine un triangle en affichant un point lorsque la valeur du tableau vaut `1` et sinon une espace.

Pour réaliser astucieusement l'initialisation de ce tableau, examinons l'emplacement des valeurs par rapport aux indices du tableau. Sachant qu'un tableau est toujours initialisé à 0 lors

Copyright © 2016 Eyrolles.

de sa création en mémoire par l'opérateur new, observons uniquement les indices correspondant aux valeurs égales à 1.

Ligne (i) \ Colonne(j)	[0]	[1]	[2]	[3]	[4]
[0]	0	0	1	0	0
[1]	0	1	1	1	0
[2]	1	1	1	1	1

Signalons, à la colonne [2], que toutes les valeurs sont initialisées à 1. Cette colonne correspond en réalité à la colonne du milieu du tableau. En supposant que l'ensemble des valeurs soit stocké dans un tableau nommé sapin, l'indice de cette colonne est obtenu grâce à l'instruction :

```
int [][] sapin = new int [3][5] ;
int milieu = sapin[0].length / 2;
```

L'expression sapin[0].length correspondant au nombre de colonnes, soit 5, la variable milieu prend pour valeur 5/2, soit 2 en entier.

Ensuite, les valeurs situées de part et d'autre de cette colonne sont elles aussi, initialisées à 1. Pour la ligne numéro [1], seul **un** élément à droite et à gauche du milieu, est initialisé à 1. Pour la ligne numéro [2], **deux** éléments à droite et à gauche, valent 1.

Il y a donc corrélation entre le nombre de valeurs à initialiser et le numéro de la ligne sur laquelle l'initialisation est effectuée. C'est pourquoi le traitement se réalise de la façon suivante :

```
for ( int i = 0 ; i < sapin.length ; i++)  {
  for ( int j = -i; j <= i; j++) {
    sapin[i][milieu+j] = 1;
  }
}
```

La variable i partant de 0 jusqu'à sapin.length (soit 3) examine toutes les lignes du tableau. Pour chaque ligne, grâce à la seconde boucle en j, les valeurs du tableau sont initialisées à 1, de part et d'autre du milieu. Pour mieux comprendre le déroulement des opérations, examinez le tableau d'évolution des variables.

i	j	milieu	sapin[i][milieu+j]
0	0	2	sapin[0][0 + 2] = 1
0	1	2	// sortie de boucle
1	-1	2	sapin[1][-1 + 2] = 1
1	0	2	sapin[1][0 + 2] = 1
1	1	2	sapin[1][1 + 2] = 1

Copyright © 2016 Eyrolles.

i	j	milieu	sapin[i][milieu+j]
1	**2**	2	// sortie de boucle
2	-2	2	sapin[**2**][**-2** + 2] = 1
2	-1	2	sapin[**2**][**-1** + 2] = 1
2	0	2	sapin[**2**][**0** + 2] = 1
2	1	2	sapin[**2**][**1** + 2] = 1
2	2	2	sapin[**2**][**2** + 2] = 1
2	**3**	2	// sortie de boucle
3		2	// sortie de boucle

Une fois le tableau créé et initialisé en mémoire, l'affichage du dessin s'effectue en testant la valeur de chaque point du tableau. Si la valeur est nulle, une espace est affichée, sinon, un point est affiché. Traduite en Java, cette marche à suivre s'écrit à l'aide de deux boucles imbriquées, comme suit :

```
for (int i = 0; i < sapin.length; i++) {
  for (int j = 0; j < sapin[0].length; j++) {
     if(sapin[i][j] == 0) {
      System.out.print(" ");
    else
      System.out.print(".");
  }
  System.out.println();
}
```

L'indice `i` représente les lignes, tandis que `j` représente les colonnes. Grâce aux boucles imbriquées, chaque intersection des lignes et colonnes est consultée, de façon à afficher le caractère correspondant à la valeur stockée. Lorsque la boucle `j` est terminée, cela signifie que tous les éléments (colonnes) de la ligne `i` ont été affichés, et il est nécessaire de passer à la ligne suivante de l'écran, grâce à l'instruction `System.out.println()`.

Créer un triangle composé de n lignes

Nous avons créé un triangle à 3 lignes composé de 5 colonnes. Si nous souhaitons ajouter une nouvelle ligne, nous devons obligatoirement ajouter deux colonnes supplémentaires. La relation entre le nombre de lignes et de colonnes s'exprime selon l'équation :

```
Nombre de colonnes = 2 * Nombre de lignes - 1
```

Pour un triangle possédant `3` lignes, vous obtenez `2 * 3 - 1 = 5` colonnes. Pour un triangle composé de `4` lignes, vous obtenez `2 * 4 - 1 = 7` colonnes. L'équation reste valide pour un triangle à une ligne, puisque le nombre de colonnes vaut `2 * 1 - 1 = 1`.

Copyright © 2016 Eyrolles.

Les instructions suivantes permettent de créer un triangle dont le nombre de lignes est déterminé par l'utilisateur :

```
import java.util.*;
public class Sapin {
 public static void main(String  [] arg)  {
  Scanner lectureClavier = new Scanner(System.in);
  System.out.print("Nombre de lignes :  ");
  int nl = lectureClavier.nextInt();
  if (nl <= 0) {
   System.out.println("Le nombre de lignes doit être
                       superieur a 0 ");
   System.exit(0);
  }
  int nc = 2*nl-1;
  int [][] sapin = new int[nl][nc];
  int milieu = sapin[0].length/2;
  for ( int i = 0 ; i < nl ; i++) {
   for ( int j = -i; j <= i; j++) {
     sapin[i][milieu+j] = 1;
   }
  }
 } // Fin de la fonction main()
} // Fin de la classe Sapin
```

Placer des décorations au hasard

Cela fait, nous sommes en mesure d'afficher le sapin sans décoration puisque, pour chaque élément du tableau valant 1, un point est affiché. Pour ajouter quelques décorations, l'idée est de placer au hasard d'autres valeurs que 1. Ainsi, l'affichage peut être modulé en fonction de la valeur rencontrée.

Pour placer dans notre sapin de nouvelles valeurs au hasard, comprises entre 1 et 6, par exemple, il suffit de modifier l'initialisation du tableau de la façon suivante :

```
  for ( int i = 0 ; i < nl ; i++)    {
   for ( int j = -i; j <= i; j++)    {
    sapin[i][milieu + j] = (int ) (5 * Math.random() + 1);
```

L'affichage du sapin se déroule ensuite comme suit :

```
  for (int i = 0; i < sapin.length; i++) {
    for (int j = 0; j < sapin[0].length; j++) {
     switch (sapin[i][j]) {
      case 0 : System.out.print(" ");
      break;
```

Copyright © 2016 Eyrolles.

```
            case 2 : System.out.print("%");
            break;
            default : System.out.print(".");
          }
        }
        System.out.println();
    }
```

Suivant la valeur contenue en `sapin[i][j]`, le programme affiche une espace, un point ou un `%`. Observez que les valeurs `1`, `3`, `4` et `5` affichent toutes un point. Seule la valeur `2` permet l'affichage d'une guirlande. Ce choix a pour effet d'afficher volontairement plus de points que de guirlandes de façon à obtenir un sapin qui ne soit pas trop surchargé.

Question Comment placer une nouvelle décoration à ce sapin ?

Réponse Il suffit d'ajouter une nouvelle option de traitement, comme par exemple :

```
case 5 : System.out.print("!");
break;
```

Attention aux boucles imbriquées

Pour manipuler des tableaux à deux dimensions, le programmeur utilise deux boucles `for` imbriquées. Dans ce cas, une boucle `for` est placée à l'intérieur d'une première boucle `for`. Ce type d'écriture nécessite attention, car certaines erreurs peuvent empêcher le bon déroulement du programme.

Une erreur d'inattention commise, en particulier, à cause des facilités du copier-coller peut aboutir à programmer deux boucles imbriquées qui utilisent la même variable comme compteur de boucles. Ainsi, en écrivant :

BOUCLES IMBRIQUÉES (i EST COMPTEUR DE BOUCLES)	VARIATION DE LA VARIABLE i
`int i;`	Boucle 1 : i = 1, i <= 3
`for(i = 1; i <= 3; i= i+1) {`	Boucle 2 : i = 1, i <= 4
`// Boucle 1`	Boucle 2 : i = 2, i <= 4
`for(i = 1; i <= 4; i= i+1) {`	Boucle 2 : i = 3, i <= 4
`// Boucle 2`	Boucle 2 : i = 4, i <= 4
`}`	Boucle 2 : **i = 5, i > 4**
`}`	Boucle 1 : **i = 6, i > 3**

Copyright © 2016 Eyrolles.

Le compteur de boucles `i` est déclaré à l'extérieur des boucles. Les deux boucles utilisent la même case mémoire pour stocker les variations de la valeur de `i`.

En entrant dans la boucle `1`, `i` prend la valeur `1`, de même qu'en entrant dans la boucle `2`. Puis `i` est incrémenté de `1` à chaque tour de la boucle `2`, jusqu'à ce que `i` dépasse la valeur `4`. La `boucle 2` est alors terminée, `i` vaut par conséquent `5`. On entre à nouveau dans la boucle `1`, `i` est incrémenté de `1` (`i` vaut `6`) puis testé. `6` étant supérieur à `3`, la boucle `1` est terminée. La boucle `1` n'est donc parcourue qu'une seule fois au lieu de trois.

Remarques

Si le compteur de boucles est déclarée à l'intérieur de la boucle, comme suit :

```
for(int i = 1; i <= 3; i= i+1) {
// Boucle 1
  for( int i = 1; i <= 4; i= i+1) {
  // Boucle 2
  }
}
```

alors, le compilateur détecte une erreur du type : `Variable 'i' is already defined in this method`. En effet, le fait de déclarer un compteur de boucles portant le même nom, dans chaque boucle, revient à déclarer, dans un même bloc de programme, deux variables portant le même nom.

L'écriture de deux boucles non imbriquées utilisant la même variable comme compteur de boucles n'est pas une erreur.

```
for( int i = 1; i <= 3; i= i+1) {
  // Premier for, 3 tours
}
for( int i = 1; i <= 6; i= i+1) {
  // Deuxième for, 6 tours
}
```

L'emploi d'une même variable de compteur pour deux boucles disjointes est correct. En effet, la variable `i` est déclarée dans la boucle. Elle n'existe en mémoire que le temps d'utilisation de la boucle. Lorsque `i` est à nouveau déclarée dans la deuxième boucle, la variable `i` précédente n'existe déjà plus. Cette manière de programmer est courante, car elle facilite la lecture des boucles. Très souvent, les compteurs de boucles ont pour nom `i`, `j` ou `k`.

Copyright © 2016 Eyrolles.

Résumé

Les tableaux sont utilisés pour regrouper sous un même nom de variable un nombre donné de valeurs de même type. Un tableau est défini par :

- un nom ;
- un type ;
- le nombre de dimensions ;
- une taille pour chacune des dimensions.

Déclaration d'un tableau

Comme toute variable, un tableau doit être déclaré. La syntaxe est la suivante :

- Pour un tableau à **une dimension** :

```
float [] donnee  = new float [5] ;
```

Cette instruction déclare un tableau composé de nombres réels de simple précision, appelé `donnee`. L'opérateur `new` réserve 5 cases mémoire de 4 octets chacune.

- Pour un tableau à **deux dimensions** :

```
int [][] valeur  = new int [3][2] ;
```

Cette instruction déclare un tableau à deux dimensions, composé de nombres entiers, appelé `valeur`. L'opérateur `new` réserve `3 * 2 = 6` cases mémoire de `4` octets chacune.

La **taille** d'un tableau est une valeur entière définie soit à l'intérieur du programme, soit saisie au clavier lors de l'exécution du programme. Une fois la taille fixée par l'opérateur `new`, il n'est plus possible de la modifier en cours d'exécution.

Un tableau n'est pas nécessairement de type simple (`int`, `double`, etc.). Il peut être de type structuré (`String` ou type défini par le programmeur, etc.). Dans ce cas, le tableau est un tableau d'objets stockant dans chacune de ses cases l'adresse d'un objet à mémoriser.

Pour accéder à une case (élément) du tableau, il suffit de placer, derrière le nom du tableau, le numéro de la case (**indice**) entre `[]`. Chaque indice est une expression entière. La première valeur d'un tableau est stockée à l'indice `0` du tableau et non à l'indice `1`.

```
for (int i = 0; i < donnee.length; i++)
  System.out.println(" " + donnee[i]);
```

Ainsi, la boucle `for` ci-dessus permet d'accéder à chaque élément du tableau. Pour ne pas dépasser la taille du tableau, il est conseillé d'utiliser la donnée `length`, qui correspond à la longueur du tableau. Le programme dépasse la taille du tableau lorsque la valeur de l'indice est supérieure à la taille déclarée du tableau. L'interpréteur détecte alors une erreur du type : `java.lang.ArrayIndexOutOfBoundsException`.

La version 1.5 du langage Java offre une version simplifiée de la boucle `for` pour parcourir un tableau de bout en bout. La nouvelle syntaxe s'écrit sous la forme suivante :

```
for (int valeur : tableau)
  System.out.println(" " + valeur);
```

Copyright © 2016 Eyrolles.

Exercices

Les tableaux à une dimension

Exercice 9.1 Qu'affiche le programme suivant ?

```
int i ;
int []  valeur = new int[6] ;
valeur [0] = 1;

for (i = 1; i < valeur.length; i++)
 valeur[i] = valeur[i-1]+2;
for (i = 0; i < valeur.length; i++)
  System.out.print("valeur["+i+"] = " + valeur[i]);
for (int v : valeur)
 System.out.print("valeur = " + v);
```

Quelles différences constatez-vous dans l'utilisation des deux dernières boucles ? Quelle instruction faudrait-il ajouter pour obtenir exactement le même affichage ?

Exercice 9.2 Écrivez un programme qui :

a. Stocke dans un tableau des valeurs entières passées en paramètres de la ligne de commande.

b. Calcule la somme de ces valeurs.

c. Calcule la moyenne de ces valeurs.

d. Recherche la plus grande valeur du tableau.

e. Détermine la position de la plus grande valeur.

f. Affiche le nombre de valeurs supérieures à la moyenne.

Exercice 9.3 Dans la classe `Triangle` réalisée au cours de l'exercice 8.8 du chapitre précédent :

a. Définir les sommets d'un triangle à l'aide de deux tableaux de 3 entiers. Le premier tableau nommé `xPoints` stocke les coordonnées en X des trois sommets du triangle, le second nommé `yPoints` stocke les coordonnées en Y.

b. Modifier les deux constructeurs de façon à enregistrer les coordonnées des sommets dans les deux tableaux. L'indice `0` de chacun des deux tableaux contient les coordonnées du point de référence défini dans la classe `Forme`, l'indice `1` les coordonnées du deuxième sommet et l'indice `2` les coordonnées du troisième sommet.

c. Modifier les méthode `afficher()` et `deplacer()` en tenant compte de la nouvelle représentation des propriétés.

Copyright © 2016 Eyrolles.

Les tableaux d'objets

Exercice 9.4 En reprenant la classe `Cercle` définie au chapitre 8 « Les principes du concept d'objet », écrivez un programme qui :

a. Crée un tableau de type `Cercle`, dont la taille est choisie par l'utilisateur. Si le nombre de cercles créés est inférieur à 4, le programme initialise par défaut la taille du tableau à 4.

b. Initialise les données de chaque tableau à l'aide du constructeur par défaut de la classe `Cercle`.

c. Déplace le cercle n° 1 en 20, 20.

d. Agrandit le cercle n° 2 de 50.

e. Échange le cercle n° 0 avec le n° 3.

f. Permute les cercles, de façon à ce que le cercle 0 soit stocké en 1, le cercle 1 en 2... et le cercle 3 en 0.

Les tableaux à deux dimensions

Exercice 9.5 Écrivez un programme qui :

a. À l'aide de boucles imbriquées, initialise la matrice 7 * 7 aux valeurs suivantes :

```
1 0 0 1 0 0 1
0 1 0 1 0 1 0
0 0 1 1 1 0 0
1 1 1 1 1 1 1
0 0 1 1 1 0 0
0 1 0 1 0 1 0
1 0 0 1 0 0 1
```

b. Affiche à l'écran le tableau, en remplaçant les valeurs :

0 par un espace (" ") ;

1 par un astéxrisque ("*").

Remarque Les compteurs de boucles seront astucieusement choisis afin d'initialiser automatiquement le tableau.

Copyright © 2016 Eyrolles.

Pour mieux comprendre le mécanisme des boucles imbriquées for–for

Exercice 9.6 Afin d'exécuter le programme suivant :

```
import java.util.*;
public class Exercice6 {
 public static void main (String [] parametre) {
  int i,j, N = 5;
  char C;
  Scanner lectureClavier = new Scanner(System.in);
  System.out.print("Entrer un  caractere :");
  C = lectureClavier.next().charAt(0);
  for (i = 1; i < N; i++)  {
   for (j = 1; j < N; j++)  {
    if (i < j) System.out.print(C);
    else System.out.print(" ");
   }
  }
 }
}
```

a. Examinez le code source, repérez les instructions concernées par les deux boucles répétitives, et déterminez les instructions de début et de fin de boucle.

b. Quelles sont les instructions qui permettent de modifier le résultat du test de sortie de boucle ?

c. En supposant que l'utilisateur entre la valeur « ! », exécutez le programme suivant à la main (pour vous aider, construisez le tableau d'évolution de chaque variable déclarée).

d. Quel est le résultat affiché à l'écran ?

Exercice 9.7 En construisant le tableau d'évolution de la variable `i`, que constatez-vous lors de l'exécution de ces boucles ?

```
for(i = 1; i <= 5; i = i+1)
{
  for(i = 1; i <= 2; i= i+1)
  {
    System.out.print("i = "+i);
  }
}
```

Copyright © 2016 Eyrolles.

Le projet : Gestion d'un compte bancaire

Traiter dix lignes comptables

L'objectif est de traiter, non plus une seule ligne comptable, mais dix lignes comptables. Pour cela, vous devez, dans un premier temps, modifier la déclaration de la donnée `ligne`, dans la classe `Compte`, comme suit :

```
private LigneComptable [] ligne;
```

Comme le nombre de lignes comptables est fixé dans le cahier des charges, il est possible de définir une constante comme suit :

```
public static final int NBLigne = 10 ;
```

`NBLigne` représente le nombre maximal de lignes comptables à traiter. Les lignes comptables étant créées au fur et à mesure des opérations réalisées par l'utilisateur, il est nécessaire de définir une variable (`nbLigneRéel`), qui compte le nombre de lignes comptables effectivement créées en cours d'exécution du programme.

La gestion des lignes comptables entraîne la modification des méthodes `Compte()`, `créerLigne()` et `afficherCompte()`.

Transformer les constructeurs Compte()

Dans chaque constructeur :

a. À l'aide de l'opérateur `new`, créer en mémoire la donnée `ligne`, sous forme d'un tableau de dix lignes comptables.

b. Initialiser la variable `nbLigneRéel` à `-1`, puisqu'aucune ligne n'a encore été saisie.

Transformer la méthode créerLigne()

Lorsque le nombre de lignes comptables traité est supérieur à 10, le programme doit effacer la première ligne traitée, de façon à décaler les suivantes (la deuxième allant en première position, la troisième en deuxième position, etc.) afin de pouvoir stocker la nouvelle ligne en dernière position du tableau `ligne`.

La méthode `créerLigne()` réalise ce traitement de la façon suivante :

a. Incrémente `nbLigneRéel` de `1`.

b. Si le nombre de lignes créées est inférieur à `NBLigne`, crée en mémoire une ligne comptable grâce au constructeur de la classe `LigneComptable` et stocke en mémoire son adresse dans le tableau `ligne`.

c. Si le nombre de lignes est supérieur à `NBLigne`, décale toutes les lignes vers le haut, grâce à la méthode `décalerLesLignes()` décrite ci-dessous, et stocke la nouvelle ligne comptable en dernière position (`NBLigne - 1`) du tableau `ligne`.

Copyright © 2016 Eyrolles.

```
private void décalerLesLignes() {
 for(int i = 1; i < NBLigne ; i++)
     ligne[i-1] = ligne[i];
}
```

d. Modifie la valeur courante du compte en fonction du crédit ou débit réalisé par la nouvelle ligne comptable.

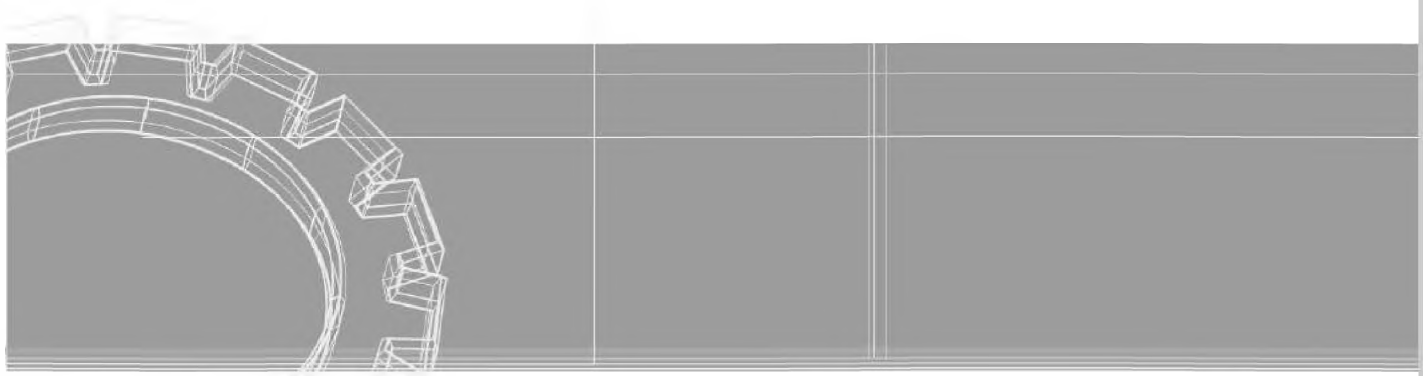
Transformer la méthode afficherCompte()

Modifier la méthode `afficherCompte()` de façon à afficher l'ensemble des lignes saisies en cours d'exécution du programme.

Copyright © 2016 Eyrolles.

Chapitre 10

Collectionner un nombre indéterminé d'objets



Comme nous l'avons vu au cours du chapitre précédent, les tableaux permettent la manipulation rapide et efficace d'un ensemble de données. Cependant, leur principal inconvénient est d'être de taille fixe. Ainsi, l'ajout d'un élément dans un tableau demande une gestion rigoureuse des indices afin d'éviter que ces derniers ne prennent une valeur supérieure à la taille du tableau.

Pour pallier cette difficulté majeure pour un grand nombre de programmes, le langage Java propose plusieurs outils de manipulation des données en mémoire vive, au fur et à mesure des besoins de l'application. Ces outils sont présentés et analysés à la section « La programmation dynamique ».

En outre, lorsqu'un programme utilise des collections importantes de données, il doit les archiver de façon à ne pas les voir disparaître après l'arrêt de l'application ou de l'ordinateur. Le langage Java offre différentes méthodes pour réaliser ce stockage de données. Elles sont étudiées à la section « L'archivage de données ».

La programmation dynamique

À la différence de la programmation **statique**, dans laquelle le nombre de données géré par l'application est fixé une fois pour toutes lors de l'exécution du programme, la programmation **dynamique** offre l'avantage de gérer un nombre indéterminé d'objets, en réservant des espaces mémoire, au fur et à mesure des besoins de l'utilisateur.

Copyright © 2016 Eyrolles.

Cette technique se montre très utile lorsque le nombre d'objets à traiter n'est pas connu ni définissable avant l'exécution du programme. Par exemple, tous les logiciels de gestion, et c'est une grande part des programmes informatiques, se doivent de gérer les données qu'ils traitent de façon dynamique.

En effet, sans programmation dynamique, vous pourriez voir une bibliothèque refuser de nouveaux lecteurs sous prétexte que le logiciel qu'elle utilise ne serait pas en mesure de traiter plus de 50 000 inscriptions, ou encore voir un logiciel de traitement de texte s'interrompre parce qu'il lui serait impossible de gérer la saisie et l'affichage de plus 10 000 caractères.

Pour éviter de telles situations, le langage Java propose différents outils qui gèrent dynamiquement les données d'un programme. En particulier, il existe des objets de type `ArrayList`, dont nous analysons les caractéristiques à la section « Les listes ». Les objets de type `HashMap`, étudiés à la section « Les dictionnaires », offrent aussi l'avantage de gérer les données de façon dynamique, tout en organisant l'information de façon à faciliter son exploitation.

Les listes

Les listes sont des objets de type `ArrayList`, un type prédéfini du langage Java. La gestion des listes est assez similaire à la gestion d'un tableau puisque le programme crée une liste par ajout de données au fur et à mesure des besoins de l'utilisateur. Les données sont enregistrées dans leur ordre d'arrivée. Un indice géré par l'interpréteur permet de retrouver l'information.

Les données enregistrées dans une `ArrayList` sont en réalité rangées dans un tableau interne créé par l'interpréteur. La taille du tableau interne est gérée automatiquement par Java. Ainsi, lorsque la liste des éléments à ajouter dépasse la taille du tableau interne, un nouveau tableau est créé et les anciennes valeurs y sont copiées.

Manipulation d'une liste

Pour utiliser une liste, il est nécessaire de la déclarer de la façon suivante :

```
ArrayListliste = new ArrayList () ;
```

Ainsi déclaré, `liste` est un objet de type `ArrayList`, auquel on peut appliquer des méthodes de la classe `ArrayList`. Ces méthodes, décrites dans le tableau ci-après, permettent l'ajout, la suppression ou la modification d'une donnée dans la liste.

Remarque

Il est possible de construire une liste en précisant la capacité de stockage du tableau interne. De cette façon, on évite la répétition des opérations de création de tableaux internes et de recopie de valeurs qui sont des opérations coûteuses en temps d'exécution. La création d'une liste de taille connue s'effectue en indiquant au constructeur le nombre estimé d'éléments contenus dans liste :

```
int capacitéInitiale = 20 ;
ArrayList liste = new ArrayList(capacitéInitiale);
```

Copyright © 2016 Eyrolles.

Opération	Fonction Java
Ajoute un élément `objet` en fin de liste.	`add(objet)`
Insère un élément `objet` dans la liste, à l'`indice` spécifié en paramètre.	`add(indice, objet)`
Retourne l'élément stocké à l'`indice` spécifié en paramètre.	`get(indice)`
Supprime tous les éléments de la liste.	`clear()`
Retourne l'indice dans la liste du premier `objet` donné en paramètre, ou `-1` si `objet` n'existe pas dans la liste.	`indexOf(objet)`
Retourne l'indice dans la liste du dernier `objet` donné en paramètre, ou `-1` si `objet` n'existe pas dans la liste.	`lastIndexOf()`
Supprime l'`objet` dont l'`indice` est spécifié en paramètre.	`remove(indice)`
Supprime tous les éléments compris entre les indices `i` (valeur comprise) et `j` (valeur non comprise).	`removeRange(i, j)`
Remplace l'élément situé en position `i` par l'`objet` spécifié en paramètre.	`set(objet, i)`
Retourne le nombre d'éléments placés dans la liste.	`size()`

Exemple : créer un nombre indéterminé d'étudiants

Pour mieux comprendre l'utilisation des listes, reprenons l'exemple de la classe d'étudiants, de façon à ce que le programme traite, non plus un nombre fixe d'étudiants, mais un nombre indéterminé.

Pour en savoir plus Sur les classes `Etudiant` et `Cursus`, voir au chapitre 9, « Collectionner un nombre fixe d'objets », la section « Trier un ensemble de données ».

La classe `Etudiant` n'est pas à modifier, puisque l'objectif est de transformer uniquement la gestion en mémoire des étudiants. La correction porte donc sur la classe `Cursus`, où la `liste` d'étudiants doit être déclarée de type `ArrayList` au lieu d'être déclarée sous forme de tableau. Examinons la nouvelle classe `Cursus` :

```
import java.util.*;
public class Cursus {
private ArrayList liste;
```

Copyright © 2016 Eyrolles.

```
 public Cursus()   {
  liste = new ArrayList ();
 }
 public void ajouteUnEtudiant() {
  liste.add(new Etudiant());
 }
public void afficheLesEtudiants() {
  int nbEtudiants = liste.size();
  if (nbEtudiants > 0) {
    Etudiant tmp;
    for (int i = 0; i < nbEtudiants; i ++) {
      tmp = (Etudiant) liste.get(i);
      tmp.afficheUnEtudiant();
    }
  }
  else System.out.println("Il n'y a pas d'etudiant dans cette
liste");
 }
} // Fin de Cursus
```

Remarque

La boucle `for` permettant l'affichage des étudiants contenus dans la liste peut s'écrire plus simplement en utilisant la nouvelle syntaxe de la version 1.5 du JDK. Elle nécessite cependant l'utilisation de type « génériques » qui sont également une nouvelle fonctionnalité de la dernière version du compilateur. Pour plus de détails reportez-vous en section « Les types génériques » plus bas dans ce chapitre.

Les outils comme `ArrayListsont` proposés par le langage Java. Ils sont définis à l'intérieur de classes, qui ne sont pas, par défaut, directement accessibles par le compilateur. C'est pourquoi le programmeur doit préciser au compilateur où se situe la bibliothèque du langage Java définissant l'outil utilisé (`package`). Pour ce faire, il doit placer une instruction `import` en première ligne du fichier qui utilise l'outil souhaité.

La classe `ArrayList` étant définie dans le package `java.util`, il convient de placer l'instruction `import java.util.*;` en tête du fichier. En effet, si cette instruction fait défaut, le compilateur détecte une erreur du type : `Class ArrayList not found`.

Cela fait, la donnée `liste`, déclarée de type `ArrayList`, doit être manipulée en tant que telle dans chaque méthode de la classe. Les trois méthodes suivantes sont définies à l'intérieur de celle-ci :

- Le constructeur `Cursus()`, qui fait appel au constructeur de la classe `ArrayList` afin de déterminer l'adresse du premier élément de la `liste`.
- La méthode `ajouteUnEtudiant()`, qui place un élément dans la `liste` grâce à la méthode `add()`. L'élément ajouté à la liste est un objet de type `Etudiant`, créé par

Copyright © 2016 Eyrolles.

l'intermédiaire du constructeur `Etudiant()`, qui demande la saisie au clavier des données caractéristiques de l'étudiant à construire.

- La méthode `afficheLesEtudiants()` parcourt l'ensemble de la liste grâce à la méthode `get()`, qui fournit en résultat l'élément stocké à la position spécifiée en paramètre, soit `i`. Ce résultat, pour être consultable, doit obligatoirement être « casté » en `Etudiant`.

Pour en savoir plus Sur le mécanisme du cast, voir au chapitre 1, « Stocker une information », la section « La transformation de types ».

- En effet, une liste a la capacité de mémoriser n'importe quel type d'objet, prédéfini ou non. Il est donc nécessaire d'indiquer au compilateur à quel type correspond l'objet extrait.

  L'indice `i`, variant de `0` jusqu'à la taille effective (`nbEtudiants - 1`) de la `liste`, l'ensemble des étudiants contenus dans la `liste` est affiché.

Remarque L'ajout d'un élément dans une liste n'est possible que si l'élément est un objet. Il n'est en effet, pas possible d'ajouter une valeur de type simple telle que `int`, `float` ou `double`.

Pour en savoir plus Les types « génériques » sont détaillés plus bas, dans ce chapitre, section « Les types génériques ».

Exemple : l'application GestionCursus

Observons l'application `GestionCursus`, qui définit et utilise un objet `Cursus`.

```
import java.util.*;
public class GestionCursus  {
  public static void main(String [] argument)  {
  byte choix = 0 ;
  Scanner lectureClavier = new Scanner(System.in);
  Cursus C = new Cursus();
  do   {
    System.out.println("1. Ajoute un etudiant");
    System.out.println("2. Affiche la liste des eleves");
    System.out.println("3. Pour sortir");
    System.out.print("Votre choix : ");
    choix  = lectureClavier.nextByte();
    switch (choix) {
      case 1 :  C.ajouteUnEtudiant();
            break;
      case 2 :  C.afficheLesEtudiants();
            break;
```

Copyright © 2016 Eyrolles.

```
            case 3 :  System.exit(0);
            default :  System.out.println("Cette option n'existe pas ");
        }
      } while (choix != 3);
    }
}
```

Le nombre d'étudiants à traiter n'est pas déterminé à l'avance. C'est pourquoi l'application `GestionCursus` réalise, grâce à la mise en place d'une boucle `do...while`, la saisie des données au fur et à mesure des besoins de l'utilisateur. Le programme laisse le `choix` à l'utilisateur de saisir de nouveaux étudiants ou d'afficher ceux effectivement saisis. L'exécution de cette application a pour résultat à l'écran :

```
1. Ajoute un etudiant
2. Affiche la liste des eleves
3. Pour sortir
Votre choix : 2
Il n'y a pas d'etudiant dans cette liste
1. Ajoute un etudiant
2. Affiche la liste des eleves
3. Pour sortir
Votre choix : 1
Entrer le nom de l'etudiant : V.
Entrer le prenom de l'etudiant : Laurent
Combien de notes pour l'etudiant  Laurent V. :  2
Entrer la note  n° 1 :  15
Entrer la note  n° 2 :  13
1. Ajoute un etudiant
2. Affiche la liste des eleves
3. Pour sortir
Votre choix : 1
Entrer le nom de l'etudiant : Sébastien
Entrer le prenom de l'etudiant : V.
Combien de notes pour l'etudiant  Sébastien V. :  2
Entrer la note  n° 1 :  12
Entrer la note  n° 2 :  16
```

Copyright © 2016 Eyrolles.

```
1. Ajoute un etudiant
2. Affiche la liste des eleves
3. Pour sortir
Votre choix : 2
Les notes de Laurent V. sont :  15.0 13.0
Sa moyenne vaut  14.0
Les notes de Sébastien V. sont :  12.0 16.0
Sa moyenne vaut  14.0
1. Ajoute un etudiant
2. Affiche la liste des eleves
3. Pour sortir
Votre choix : 3
```

L'utilisation d'objets du type `ArrayList`est souple et facilite amplement la vie du programmeur lorsque ce dernier souhaite écrire une application qui gère des données de façon dynamique. Les méthodes de la classe `ArrayList`permettent aussi la recherche ou l'insertion de nouveaux éléments grâce, en particulier, à la méthode `indexOf(objet)`, qui retrouve l'indice de l'objet spécifié en paramètre dans la liste.

Cependant, lorsque la liste mémorise des objets complexes, tels que les données caractéristiques d'un étudiant, la recherche d'un étudiant particulier n'est pas simple. En effet, il est nécessaire de fournir au programme toutes les données de l'étudiant (`nom`, `prenom`, `notes` et `moyenne`), de façon à être sûr de le retrouver dans la liste. La méthode `indexOf(objet)` ne retrouve l'objet spécifié en paramètre qu'à la seule condition qu'il soit totalement identique à celui stocké dans la `liste`.

Les dictionnaires

Pour améliorer la recherche d'éléments complexes dans une liste, la technique consiste à organiser les données, non plus par rapport à un indice, mais par rapport à une clé explicite. De cette façon, la recherche d'un objet dans la liste s'effectue, non plus sur l'ensemble des données qui le composent, mais sur une clé unique qui lui est associée.

L'organisation de données, par association d'une clé à un ensemble de données, est appelée un dictionnaire.

Remarque

Dans un dictionnaire, chaque définition est associée au mot qu'elle définit et non pas à sa position (numérique) dans le dictionnaire.

Copyright © 2016 Eyrolles.

Manipulation d'un dictionnaire

Le type `HashMap` proposé par le langage Java permet de réaliser simplement l'association clé-élément. Les méthodes associées à ce type sont la création, la suppression, la consultation ou la modification d'une association.

Pour utiliser un dictionnaire, il est nécessaire de le déclarer de la façon suivante :

```
HashMap listeClassee = new HashMap() ;
```

Ainsi déclaré, `listeClassee` est un objet de type `HashMap`, sur lequel on peut appliquer des méthodes de la classe `HashMap`. Les méthodes les plus couramment utilisées sont décrites au tableau ci-après.

Opération	Fonction Java
Place dans le dictionnaire l'association `cle-objet`.	`put(cle, objet)`
Retourne l'objet associé à la `cle` spécifiée en paramètre.	`get(cle)`
Supprime dans le dictionnaire l'association `cle-objet` à partir de la `cle` spécifiée en paramètre.	`remove(cle)`
Retourne le nombre d'associations définies dans le dictionnaire.	`size()`

Exemple : créer un dictionnaire d'étudiants

Pour mieux comprendre l'utilisation de tels objets, modifions le programme de gestion d'un ensemble d'étudiants de façon à organiser les données à partir d'une clé définie par le programme.

Définir une clé d'association

En supposant qu'un étudiant soit totalement identifiable par son nom et son prénom, la clé d'association des données est définie comme étant une chaîne de caractères majuscules, dont le premier caractère coïncide avec le premier caractère du prénom de l'étudiant et dont les caractères suivants correspondent au nom de l'étudiant.

De cette façon, chaque clé est déterminée par programme, indépendamment de l'utilisateur, en fonction des informations fournies par ce dernier.

La traduction de cet algorithme en langage Java est la suivante :

```
private String creerUneCle(Etudiant e) {
  String tmp;
  tmp = (e.quelPrenom()).charAt(0) + e.quelNom();
  return tmp.toUpperCase();
 }
```

Copyright © 2016 Eyrolles.

À partir des données d'un étudiant e passées en paramètres, la méthode `creerUneCle()` retourne une chaîne de caractères majuscules (`tmp.toUpperCase()`), composée du premier caractère du prénom de l'étudiant (`(e.quelPrenom()).charAt(0)`), suivi de son nom (`e.quelNom()`).

Remarque Les données `nom` et `prenom` de la classe `Etudiant` sont privées. Il est donc nécessaire d'utiliser les méthodes d'accès en consultation `quelPrenom()` et `quelNom()` pour connaître le contenu de ces données.

Ces méthodes, **à insérer dans le fichier** `Etudiant.java`, sont décrites comme suit :

```
public String quelNom()  {
  return nom;
}
public String quelPrenom()  {
  return prenom;
}
```

Question La création d'une clé peut également être réalisée simplement à partir des nom et prénom de l'étudiant, stockés non pas dans un objet `Etudiant`, mais dans deux `String` distincts. Comment surcharger la méthode `creerUneCle()` de façon à traiter cette alternative ?

Réponse La méthode est surchargée de la façon suivante :

```
private String creerUneCle(String p, String n) {
  String tmp;
  tmp = p.charAt(0)+ n;
  return tmp.toUpperCase();
}
```

Les deux méthodes `creerUneCle()` **à insérer dans la classe** `Cursus`, sont déclarées en mode `private` car elles constituent un traitement interne propre au mode de stockage de l'information. L'application et l'utilisateur n'ont nullement besoin d'en connaître l'existence pour créer la liste des étudiants d'un cursus.

Création du dictionnaire

Pour créer le dictionnaire d'un ensemble d'étudiants, nous devons tout d'abord définir un objet de type `HashMap` puis stocker dans cet objet les étudiants, en les associant à leur clé. Ces deux opérations sont réalisées dans le programme suivant :

```
import java.util.*;
public class Cursus  {
 private HashMap listeClassee;
 public Cursus()   {
```

Copyright © 2016 Eyrolles.

```
    listeClassee = new HashMap();
  }
  public void ajouteUnEtudiant() {
    Etudiant nouveau = new Etudiant();
    String cle = creerUneCle(nouveau);
    if (listeClassee.get(cle)== null) listeClassee.put(cle, nouveau);
    else System.out.println("Cet etudiant a deja ete saisi !");
   }
  }
```

Ce programme est constitué des deux méthodes suivantes :

- Le constructeur `Cursus()`, qui fait appel au constructeur de la classe `HashMap` afin de déterminer l'adresse du premier élément de la `listeClassee`.
- `ajouteUnEtudiant()`, qui place un élément dans le dictionnaire grâce à la méthode `put(cle, nouveau)`, qui ajoute l'association `cle-nouveau` dans le dictionnaire `listeClassee`. L'objet `nouveau` est de type `Etudiant`. Il est créé par l'intermédiaire du constructeur `Etudiant()` et `cle` est aussi un objet de type `String` calculé à partir de la méthode `creerUneCle()`.

L'ajout successif de deux associations ayant la même `cle` a pour résultat de détruire la première association. C'est pourquoi il convient de tester, avant de placer le nouvel étudiant dans le dictionnaire, si ce dernier n'est pas déjà défini dans la `listeClassee`. C'est ce que réalise le test suivant :

```
if (listeClassee.get(cle) == null) listeClassee.put(cle, nouveau);
```

En effet, en testant le résultat de la méthode `get(cle)`, le programme recherche dans le dictionnaire s'il existe un étudiant associé à la `cle` calculée, correspondant à l'étudiant que l'on souhaite insérer dans la liste (`nouveau`). Si cette association n'existe pas, l'élément retourné par la méthode est `null`, et l'interpréteur ajoute la nouvelle association `cle-nouveau` dans le dictionnaire. Dans le cas contraire, le programme affiche un message précisant que l'étudiant existe déjà.

Rechercher et supprimer un élément du dictionnaire

Une fois le dictionnaire réalisé, les opérations permettant la recherche ou la suppression d'un étudiant sont décrites par les méthodes suivantes, à insérer dans le fichier `Cursus.java` :

```
  public void rechercheUnEtudiant(String p, String n) {
    String cle = creerUneCle(p, n);
    Etudiant eClasse = (Etudiant) listeClassee.get(cle);
    if (eClasse != null) eClassé.afficheUnEtudiant();
    else System.out.println(p + " " + n + " est inconnu ! ");
  }
  public void supprimeUnEtudiant(String p, String n) {
```

Copyright © 2016 Eyrolles.

```
    String cle = creerUneCle(p, n);
    Etudiant eClasse = (Etudiant) listeClassee.get(cle);
    if (eClasse!= null)  {
      listeClassee.remove(cle);
      System.out.println(p + " " + n + " a ete supprime ");
    }
    else System.out.println(p + " " + n + " est inconnu ! ");
  }
```

Ces méthodes fonctionnent toutes deux sur le même modèle. Les nom et prénom de l'étudiant à traiter sont fournis en paramètres des méthodes afin de calculer la clé d'association. Ensuite, l'étudiant est recherché dans la liste à partir de cette clé (`get(cle)`).

S'il est trouvé, il est soit affiché (`eClasse.afficheUnEtudiant()`, pour la méthode `rechercheUnEtudiant()`), soit supprimé (`listeClassee.remove(cle)`), pour la méthode `supprimeUnEtudiant()`).

Afficher un dictionnaire

Pour afficher tous les éléments d'un dictionnaire, nous devons le parcourir élément par élément. Il existe différentes techniques pour réaliser ce parcours. Nous vous en proposons une, qui utilise un outil du langage Java, défini par la classe `Collection`.

Une collection est un outil du package `java.util`, qui facilite le parcours de listes de données, quelles qu'elles soient. Ainsi, pour le parcours dans une collection d'objets, s'effectue par l'intermédiaire d'un objet de type `Iterator` que l'on applique à la collection. La classe `Iterator` contient les méthodes `hasNext()` et `next()`. La première méthode détermine s'il existe encore des éléments dans la collection sur laquelle est appliquée l'itérateur, tandis que la seconde permet l'accès à l'élément suivant dans la collection.

La méthode `afficheLesEtudiants()` utilise cette technique pour réaliser la parcours de la `listeClassee`.

```
public void afficheLesEtudiants()   {
  if(listeClassee.size() != 0){
    Collection c = listeClassee.values();
             for(Iterator i = c.iterator() ; i.hasNext();) {
                Etudiant e = (Etudiant) i.next();
                e.afficheUnEtudiant();
    }
           }
           else
  System.out.println("Il n'y a pas d'etudiant dans cette liste");
}
```

La collection est initialisée grâce à la méthode `values()` de la classe `HashMap`, qui renvoie sous forme de collection la liste des associations clé-valeur effectivement stockées. Le parcours de cette collection est ensuite réalisé à l'aide d'une boucle `for` et d'un itérateur

Copyright © 2016 Eyrolles.

associé à la collection (`Iterator i = c.iterator()`). La boucle teste s'il existe encore des clés dans la liste (`i.hasNext()`). Si tel est le cas, le programme passe à l'élément suivant dans la liste (`i.next()`) et l'affiche (`e.afficheUnEtudiant()`).

Les types génériques

Les types génériques ont été mis en place avec la nouvelle version du compilateur pour offrir en tout premier lieu la possibilité de décrire des comportements identiques indépendamment du type de données. L'apport des « generiques » permet également de rendre le code plus robuste et simplifie grandement la programmation.

Notre objectif n'est pas ici de décrire toutes les fonctionnalités des types génériques, mais d'examiner comment ces derniers simplifient la manipulation des collections d'objets (`ArrayList` et `HashMap`) et permettent d'éviter des erreurs lors de l'exécution des applications manipulant des collections de données.

Les collections stockent tout type d'objets

Comme nous avons pu le constater au cours de la section « Les listes » l'ajout de valeurs à l'intérieur d'une liste ou d'un dictionnaire ne peut s'effectuer qu'au travers d'objets. Il est impossible d'insérer un simple entier dans une liste comme par exemple :

```
ArrayList listeValeur = new ArrayList();
        listeValeur.add(100);
```

Ici, l'ajout de la valeur numérique `100` provoque une erreur de compilation ayant pour message « `The method add (int, Object) in the type ArrayList is not applicable for the arguments (int)` ».

Pour insérer une valeur numérique, il convient de forcer le type simple (`int`, `char`, `double`, etc.) à devenir un objet comme le montre l'instruction suivante :

```
listeValeur.add(new Integer(100));
```

Dans cette situation, la valeur numérique `100` est transformée en un « objet » contenant la valeur `100`. Cette dernière peut alors être insérée dans une liste de type `ArrayList`.

Cela fait, rien ne nous interdit d'insérer dans la même liste `listeValeur` une nouvelle valeur comme suit :

```
listeValeur.add("22");
```

La chaîne de caractères `"22"` n'est pas un nombre mais un objet de type `String`. Il peut donc être inséré dans la liste `listeValeu` sans qu'aucune erreur ne soit détectée lors de la compilation. La liste contient deux objets de types différents.

Cependant, un problème se pose lors de la consultation de la liste. En effet, pour connaître la valeur enregistrée dans une liste il convient de connaître son type. Pour récupérer la valeur `100` nous devons écrire :

```
Integer valeur1 = (Integer) listeValeur.get(0);
```

Copyright © 2016 Eyrolles.

Mais il n'est pas possible de récupérer la valeur `"22"` de la même façon. En effet, écrire

```
Integer valeur2 = (Integer) listeValeur.get(1);
```

provoque une erreur d'exécution ayant pour message : « `Exception in thread "main" java.lang.ClassCastException: java.lang.String` ». L'interpréteur ne peut transformer une chaîne de caractères en un entier. La solution consistant à transformer la valeur `"22"` en `String` est valide. Mais elle oblige à connaître avant consultation, le type de chaque donnée enregistrée, pour chaque indice. Ce qui peut devenir très vite complexe à gérer.

Les génerics forcent le contrôle du type de données

Pour éviter ce type d'erreur, la solution consiste forcer le type de la liste à une seule forme de données en utilisant les types génériques. Grâce à cette nouvelle structure, les listes ou encore les dictionnaires sont déclarés comme étant des listes d'entiers ou de chaînes de caractères ou encore d'étudiants. Ils ne peuvent contenir aucune autre forme de données que celles spécifiées lors de la déclaration.

La structure de déclaration d'une liste utilisant des types génériques s'écrit :

```
ArrayList<Integer> listeValeur = new ArrayList<Integer>();
```

si la liste traitée ne doit contenir que des entiers, ou encore :

```
ArrayList<Etudiant> listeEtudiant = new ArrayList<Etudiant>();
```

pour définir une liste ne contenant que des objets de type `Etudiant`.

Le terme `<type>` permet d'indiquer au compilateur quel type de données peut être traité par la liste ainsi créée. Il se traduit en français par « l'objet `listeValeur` est une liste ne contenant que des entiers » ou encore « l'objet `listeEtudiant` est une liste ne contenant que des étudiants ».

De cette façon, une instruction du type :

```
listeValeur.add("22");
```

entraîne une erreur décelable dès la phase de compilation. En effet, `listeValeur` grâce au générique `<Integer>` ne peut contenir que des entiers et `"22"` est de type `String`. L'ambiguïté est levée bien avant la phase d'exécution.

Remarque

Pour déclarer un dictionnaire d'étudiants à l'aide des types génériques, il convient d'écrire :

```
HashMap<String, Etudiant> listeClassee =
                                new HashMap<String, Etudiant> ();
```

Le premier terme, placé entre < >, indique que la clé d'association est de type `String`, alors que le second définit le type des données (élément) enregistrées dans le dictionnaire.

Copyright © 2016 Eyrolles.

Les génériques simplifient le parcours des listes

Grâce aux types génériques et à la nouvelle syntaxe de la boucle `for`, l'affichage des éléments d'une liste ou d'un dictionnaire est simplifié. Ainsi avant la version 1.5 du compilateur nous devions pour parcourir une liste d'étudiants, écrire les instructions suivantes :

```
Etudiant tmp;
❶ for (int i = 0; i < nbEtudiants; i++) {
❷  tmp = (Etudiant)liste.get(i);
tmp.afficheUnEtudiant();
}
```

❶ La boucle `for` utilise un compteur, permettant le parcours de la liste élément par élément, à partir de leur indice d'enregistrement.

❷ Lors de la consultation de la liste, il est obligatoire d'utiliser les mécanismes du « cast » pour s'assurer que l'objet extrait à l'aide de la méthode `get()` soit du type attendu.

Avec les types génériques il n'est plus besoin d'utiliser le mécanisme du « cast », et la boucle `for` s'écrit beaucoup plus simplement comme le montre les instructions ci-après :

```
ArrayList<Etudiant> listeEtudiant = new ArrayList<Etudiant> ();
for (Etudiant e : listeEtudiant) e.afficheUnEtudiant();
```

La boucle `for` se traduit littéralement de la façon suivante : « pour chaque étudiant `e` contenu dans la liste `listeEtudiant`, afficher son contenu ».

De la même façon, il n'est plus besoin d'utiliser un `Iterator` pour parcourir un dictionnaire. Ainsi, avec les instructions :

```
❶ HashMap<String, Etudiant> dico = new HashMap<String, Etudiant> ();
❷ Collection<Etudiant> lc = dico.values();
❸ for ( Etudiant e : lc) e.afficheUnEtudiant();
```

Le parcours du dictionnaire s'effectue à partir d'une collection d'étudiants (❷) grâce à l'utilisation du type générique `<Etudiant>`. Cette opération est valide , l'objet `dico` ne contenant que des objets de type `Etudiant` et des clés d'association de type `String` (❶).

Pour finir, la boucle `for` (❸) se traduit par l'expression : « pour chaque étudiant `e` contenu dans la collection `lc`, afficher son contenu ».

Exemple : l'application GestionCursus

L'emploi des types génériques transforme quelque peu la classe `Cursus`. La déclaration de la propriété `listeClassee` et le constructeur de la classe s'écrivent à l'aide des instructions :

```
private HashMap<String, Etudiant> listeClassee;
 public Cursus()   {
        listeClassee = new HashMap<String, Etudiant> ();
 }
```

Copyright © 2016 Eyrolles.

La méthode permettant l'affichage de la liste des étudiants s'écrit :

```
public void afficheLesEtudiants() {
 if (listeClassee.size() != 0) {
     Collection<Etudiant> c = listeClassee.values();
     for (  Etudiant e : c)  e.afficheUnEtudiant();
 }
 else
    System.out.println("Il n'y a pas d'etudiant dans cette liste");
 }
```

Ces transformations sont à insérer dans le fichier `Cursus.java`.

Ensuite, la gestion des étudiants d'une classe est totalement achevée en écrivant l'application `GestionCursus` comme suit :

```
import java.util.*;
public class GestionCursus  {
public static void main (String [] argument) {
  byte choix = 0 ;
  Cursus C = new Cursus();
  String prenom, nom;
  Scanner lectureClavier = new Scanner(System.in);
  do  {
    System.out.println("1. Ajoute un etudiant");
    System.out.println("2. Supprime un etudiant");
    System.out.println("3. Affiche la liste des eleves");
    System.out.println("4. Affiche un etudiant");
    System.out.println("5. Sortir");
    System.out.println();
    System.out.print("Votre choix : ");
    choix  = lectureClavier.nextByte();
    switch (choix) {
      case 1 : C.ajouteUnEtudiant();
      break;
      case 2 :
            System.out.print("Entrer le prenom de l'etudiant : ");
            prenom = lectureClavier.next();
            System.out.print("Entrer le nom de l'etudiant : ");
            nom = lectureClavier.next();
            C.supprimeUnEtudiant(prenom, nom);
      break;
      case 3 :  C.afficheLesEtudiants();
      break;
```

Copyright © 2016 Eyrolles.

```
        case 4 :
              System.out.print("Entrer le prenom de l'etudiant : ");
              prenom = lectureClavier.next();
              System.out.print("Entrer le nom de l'etudiant : ");
              nom = lectureClavier.next();
              C.rechercheUnEtudiant(prenom, nom);
        break;
        case 5 :  System.exit(0) ;
        default : System.out.println("Cette option n'existe pas ");
      }
    } while ( choix != 5);
  }
}
```

Chaque option du menu utilise une méthode de la classe `Cursus`. Ces options offrent la possibilité d'ajouter, de supprimer et de consulter tout ou partie du dictionnaire, formé au fur et à mesure des choix de l'utilisateur. L'exécution de cette application peut avoir, par exemple, pour résultat à l'écran :

```
1. Ajoute un etudiant
2. Supprime un etudiant
3. Affiche la liste des eleves
4. Affiche un etudiant
5. Sortir
Votre choix : 1
Entrer le nom de l'etudiant : R.
Entrer le prenom de l'etudiant : Sylvain
Combien de notes pour l'etudiant  Sylvain R. :  2
Entrer la note  n° 1 :  15
Entrer la note  n° 2 :  14
1. Ajoute un etudiant
2. Supprime un etudiant
3. Affiche la liste des eleves
4. Affiche un etudiant
5. Sortir
Votre choix : 1
Entrer le nom de l'etudiant : C.
Entrer le prenom de l'etudiant : Gaelle
Combien de notes pour l'etudiant  Gaelle C. :  2
Entrer la note  n° 1 :  16
Entrer la note  n° 2 :  12
```

Copyright © 2016 Eyrolles.

```
1. Ajoute un etudiant
2. Supprime un etudiant
3. Affiche la liste des eleves
4. Affiche un etudiant
5. Sortir
Votre choix : 4
Entrer le prenom de l'étudiant recherche : C.
Entrer le nom de l'étudiant recherche : Gaelle
C. Gaelle est inconnu !
1. Ajoute un etudiant
2. Supprime un etudiant
3. Affiche la liste des eleves
4. Affiche un etudiant
5. Sortir
Votre choix : 4
Entrer le prenom de l'étudiant recherche : Gaelle
Entrer le nom de l'étudiant recherche : C.
Les notes  de Gaelle C. sont :  16.0 12.0
Sa moyenne vaut  14.0
1. Ajoute un etudiant
2. Supprime un etudiant
3. Affiche la liste des eleves
4. Affiche un etudiant
5. Sortir
Votre choix : 5
```

Lors du premier choix `4`, l'utilisateur a inversé les nom et prénom de l'étudiante. La clé qui en découle n'existe pas dans le dictionnaire. Le programme ne peut donc pas retrouver les informations concernant cette étudiante.

Ainsi, grâce aux objets de type `HashMap`, il est possible d'organiser, sans beaucoup d'efforts de programmation, des données de façon à pouvoir rechercher, modifier ou supprimer un élément dans une liste.

Pourtant, l'application `GestionCursus` possède encore un inconvénient majeur : elle perd la mémoire… En effet, à chaque exécution, les données doivent de nouveau être saisies au clavier. Les données stockées dans la mémoire vive de l'ordinateur se perdent à l'arrêt du programme. Pour corriger ce défaut, le programme doit pouvoir enregistrer les informations traitées dans un fichier sur le disque dur. Cet enregistrement des données est aussi appelé archivage de données.

Copyright © 2016 Eyrolles.

Les streams et les expressions lambda

Avec la version 8 de Java, apparaissent de nouvelles fonctionnalités telles que les streams et les expressions lambda. Il s'agit d'améliorations très attendues puisqu'elles apportent simplification et puissance dans l'écriture des applications Java.

Les streams

Les streams sont utilisés pour effectuer des suites de traitements à la volée, sur des éléments appartenant à une collection de données. Les traitements peuvent être des tris, recherches d'éléments, changements de formats, filtres...

Remarque Le terme stream signifie flux. Il s'agit là de transformer les collections de données sous forme de flux de données, afin qu'il soit traité rapidement, sans écriture de boucles. Attention à ne pas confondre avec les `Inputstream` ou `Outputstream` qui correspondent à des flux d'entrée ou de sortie (voir section « L'archivage de données » ci-après).

La technique la plus simple pour créer un `stream` à partir d'une collection de données est d'appliquer la méthode `stream()` à une liste comme suit :

```
ArrayList<Etudiant>  liste;
liste = new ArrayList<Etudiant>();
liste.stream();
```

L'instruction `liste.stream()` a pour seul effet de transférer les données contenues dans la liste vers un flux de données qui pourra ensuite être traité par un ensemble d'opérations. Cette instruction écrite de cette façon est inutile, puisqu'un flux ne stocke ni ne modifie les données de la source (ici `liste`).

Pour être valide et effectuer une suite de traitements intermédiaires, il convient d'appliquer au flux `liste.stream()`, une suite de méthodes correspondant à des traitements spécifiques. Les traitements possibles sont, par exemple :

- `filter()` qui filtre les données selon un ou plusieurs critères fournis en paramètre ;
- `map()` qui modifie les données à la volée ;
- `sorted()` qui trie les données en fonction de critères passés en paramètre.

Ces traitements sont considérés comme des opérations intermédiaires auxquelles une méthode finale doit être appliquée pour terminer l'ensemble des opérations. Nous pouvons, par exemple, utiliser :

- `foreach()` pour répéter une opération d'affichage des résultats issus du traitement ;
- `min()` ou `max()` pour récupérer la plus petite ou la plus grande valeur ;
- `findAny()` ou `findFirst()` pour trouver un élément correspondant à des critères passés en paramètre.

Copyright © 2016 Eyrolles.

Les instructions permettant l'enchaînement des traitements ont été pensées pour être utilisées avec des expressions lambda que nous étudions ci-après.

Les expressions lambda

Les expressions lambda font partie des plus grandes nouveautés de Java 8. Elles apportent une très grande puissance de programmation fonctionnelle en simplifiant à l'extrême l'enchaînement des instructions.

La syntaxe d'écriture des expressions lambda est la suivante :

```
(paramètres) -> instructions
```

ou

`(paramètres) -> {instructions}` quand il y a plusieurs instructions.

Une expression lambda peut être considérée comme une fonction anonyme qui a accès aux variables locales ou d'instances du code appelant. L'expression mémorise les données dans les `paramètres`. Une instruction ou suite d'instructions est ensuite appliquée aux `paramètres`. Pour mieux comprendre le fonctionnement des expressions lambda, examinons l'extrait de code suivant :

```
ArrayList<Etudiant> liste = new ArrayList<Etudiant>();
// Instructions permettant la saisie des données relatives à plusieurs étudiants
// ...
// Création d'un flux et traitements
liste.stream().filter(e -> e.quelNom().startsWith("A"))
              .forEach(e -> e.afficheUnEtudiant());
```

Une fois le flux de données créé à partir de la liste des étudiants (`liste.stream()`), une suite de traitements est effectuée dans l'ordre suivant.

- Un filtre est appliqué au flux à l'aide d'une expression lambda (`filter(e->)`). Cette dernière récupère dans une boucle itérative propre au flux de données, chaque étudiant, pour le placer dans un objet temporaire nommé ici `e`.
- Le code exécuté (`e.quelNom().startsWidth("A")`) est appliqué au paramètre `e` défini au sein du filtre. Il permet ici de récupérer l'étudiant dont le nom commence par un A. Ce code est exécuté pour chaque étudiant de la `liste`.
- Pour finir, la boucle `foreach()` récupère et parcourt la liste des étudiants issue du précédent traitement, c'est-à-dire tous les étudiants dont le nom commence par un A, pour afficher leur nom, prénom et notes (`e.afficheUnEtudiant()`).

Les streams et expressions lambda apportent une très grande puissance de traitement des données, à partir d'un code Java minimaliste. Ils restent cependant difficiles d'accès pour le lecteur débutant, du fait de leur écriture très condensée.

Copyright © 2016 Eyrolles.

L'archivage de données

La notion d'archivage de données est très importante puisque, grâce à elle, les informations traitées sont stockées sous forme de fichiers sur le disque dur. Les informations ainsi stockées perdent leur volatilité et peuvent être réutilisées après un arrêt momentané du programme ou de l'ordinateur. Pour archiver des données, le langage Java utilise le concept de flux ou en anglais, stream.

La notion de flux

Lorsque nous communiquons des informations à l'ordinateur, nous effectuons une opération d'entrée.

Pour en savoir plus Sur les opérations d'entrée et de sortie, voir le chapitre 2, « Communiquer une information ».

Cette communication utilise un flux entrant, qui est en quelque sorte la concrétisation informatique du courant électrique passant du clavier à la mémoire vive de l'ordinateur. Symétriquement, il existe un flux sortant, permettant de faire passer une information stockée en mémoire vive à l'écran de l'ordinateur.

Dans le jargon informatique, on dit que les flux reliant la mémoire vive à l'écran ou au clavier utilisent des flux standards d'entrée-sortie. De façon similaire, il existe d'autres types de flux, qui relient la mémoire vive, non plus à l'écran ou au clavier, mais au disque dur de l'ordinateur. Ce sont les flux de fichiers.

Ces flux sont aussi caractérisés par leur direction, entrante ou sortante. Un flux de fichier sortant est un flux d'écriture qui réalise la création et l'enregistrement de données dans un fichier. Symétriquement, un flux de fichier entrant est un flux de lecture qui permet l'initialisation des variables ou objets du programme en mémoire vive, grâce aux valeurs précédemment enregistrées sur le disque dur.

Ces flux sont définis à travers des objets prédéfinis du langage Java (package `java.io`). Il en existe un très grand nombre, offrant tous des outils permettant le stockage et le traitement de données sous diverses formes.

Notre objectif n'est pas de les décrire tous, même succinctement, mais de présenter concrètement deux techniques d'archivage afin d'en comprendre les différents mécanismes. C'est pourquoi notre étude porte sur les fichiers stockant l'information sous la forme de caractères (voir, ci-après, la section « Les fichiers textes »), ainsi que sur les fichiers stockant des objets (voir la section « Les fichiers d'objets »).

Copyright © 2016 Eyrolles.

Les fichiers textes

Puisqu'il existe deux façons d'accéder à un fichier (lecture ou écriture), les outils proposés par le langage Java reproduisent ces deux modes d'accès. Pour manipuler des fichiers, nous devons donc déclarer deux objets différents, qui vont nous permettre de lire ou d'écrire dans un fichier.

Ainsi, la déclaration :

```
BufferedWriter fW;
```

définit un objet `fW` de type `BufferedWriter`, utilisé pour écrire (`Writer`), c'est-à-dire enregistrer des données dans un fichier. Par contre, la déclaration :

```
BufferedReader fR;
```

définit un objet `fR` de type `BufferedReader`, utilisé pour lire (`Reader`) les données contenues dans un fichier afin de les placer dans des variables (en mémoire vive). Ces objets et les méthodes associées sont définis dans le package `java.io`. Il convient donc de placer l'instruction `import java.io.* ;` en en-tête des classes qui font appel à ces outils.

Exemple : une classe pour lire et écrire du texte

L'objectif de cet exemple est de créer une classe `Fichier` composée d'outils simplifiant la manipulation des fichiers en lecture et en écriture. Les données de cette classe définissent deux objets de type `BufferedWriter` et `BufferedReader` et d'une variable de type `char`, qui mémorise le mode de traitement utilisé (lecture ou écriture).

```
import java.io.*;
public class Fichier {
  private BufferedWriter fW;
  private BufferedReader fR;
  private char mode;
}
```

D'une façon générale, les traitements sur fichiers se déroulent en trois temps : ouverture du flux, puis traitement des données parcourant le flux et, pour finir, fermeture du flux. Chacune de ces étapes est décrite au cours des sections suivantes.

Ouverture du flux

Pour écrire ou lire dans un fichier, il nécessaire, avant tout, d'ouvrir le flux en indiquant si ce flux est entrant (lecture) ou sortant (écriture). La méthode `ouvrir()` décrite ci-dessous réalise cette opération. Elle doit être insérée dans la classe `Fichier`.

```
public void ouvrir(String nomDuFichier, String s)
            throws IOException   {
  mode = (s.toUpperCase()).charAt(0);
  if (mode == 'R' || mode == 'L')
```

Copyright © 2016 Eyrolles.

```
    fR  = new BufferedReader(new FileReader(new File(nomDuFichier)));
    else  if (mode == 'W' || mode == 'E')
    fW = new BufferedWriter(new FileWriter(new File(nomDuFichier)));
  }
```

Les deux points importants suivants sont à observer dans l'en-tête de la méthode `ouvrir()` :

- Le premier concerne le terme `throws IOException`, dont la présence est obligatoire pour toutes les méthodes qui manipulent des opérations d'entrée-sortie. Succinctement, cette clause indique au compilateur que la méthode ainsi définie est susceptible de traiter ou de propager une éventuelle erreur, du type `IOException`, qui pourrait apparaître en cours d'exécution.

Pour en savoir plus Pour plus de précisions sur la notion d'exception, voir la section « Gérer les exceptions », en fin de chapitre.

- Le second point important est relatif aux informations transmises à la méthode `ouvrir()`. Le premier paramètre spécifie le nom du fichier auquel est associé le flux, tandis que le second indique le mode d'ouverture du flux (entrant ou sortant). Ce paramètre peut prendre différentes valeurs, telles que `"Ecriture"`, `"E"`, `"Write"` ou encore `"W"`, pour le mode sortant, et `"Lecture"`, `"L"`, `"Read"` ou encore `"R"`, pour le mode entrant. Ces mots peuvent être écrits indifféremment en majuscules ou en minuscules. En effet, la variable d'instance `mode` est initialisée à partir du premier caractère (`charAt(0)`) du paramètre `s` et est automatiquement transformée en majuscules (`s.toUpperCase()`).

Cela fait, le flux est ouvert en lecture ou en écriture en fonction de la variable d'instance `mode`. Ainsi :

- Si `mode` vaut `L` ou `R`, l'ouverture du fichier est réalisée en lecture grâce à l'instruction :

```
fR = new BufferedReader(new FileReader(new File(nomDuFichier)));
```

  Cette instruction relativement déconcertante pour le programmeur débutant réalise plusieurs opérations afin de déterminer où se situe le début du fichier spécifié en paramètre.

  La première opération `new File(nomDuFichier)` permet d'obtenir une représentation logique du fichier (existe-t-il ou non sur le disque dur ?). Ensuite, l'appel au constructeur `FileReader()` permet l'ouverture du fichier en lecture caractère par caractère. Il fournit en retour l'adresse du début du fichier.

  Cependant, ce mode de lecture n'autorise pas la lecture de plusieurs caractères à la fois. C'est pourquoi il est nécessaire de faire appel à un troisième constructeur, `BufferedReader()`, qui permet la lecture du fichier ligne par ligne. L'adresse du début du fichier est alors mémorisée, grâce au signe d'affectation, dans l'objet `fR`.

- Si `mode` vaut `E` ou `W`, l'ouverture du fichier est réalisée en écriture grâce à l'instruction :

```
fW = new BufferedWriter(new FileWriter(new File(nomDuFichier)));
```

Copyright © 2016 Eyrolles.

Les opérations réalisées sont équivalentes à celles décrites ci-dessus, en remplaçant le mode lecture par le mode écriture. Cependant,

- Si le fichier spécifié en paramètre n'existe pas, et :
  - Si le chemin d'accès à ce fichier dans l'arborescence du disque est valide, alors le fichier est créé, et l'adresse du début du fichier est stockée dans l'objet `fW`.
  - Si le chemin d'accès n'est pas valide, le fichier n'est pas créé, et une erreur du type `FileNotFoundException` est détectée.
- Si le fichier existe, il est ouvert, et son contenu est totalement effacé. L'adresse du début du fichier est alors stockée dans l'objet `fW`.

Traitement du fichier

Une fois le fichier ouvert, les traitements réalisables sur lui sont l'écriture et la lecture de données dans le fichier.

- **L'écriture** dans un fichier est réalisée par la méthode suivante :

```
public void ecrire(int tmp) throws IOException {
  String chaine = "";
  chaine = String.valueOf(tmp);
  if (chaine != null)   {
    fW.write(chaine,0,chaine.length());
    fW.newLine();
  }
}
```

La méthode `ecrire()` prend en paramètre la valeur à enregistrer dans le fichier. Comme il s'agit d'un entier et que le fichier est un fichier texte, la valeur stockée dans `tmp` est convertie en `String` grâce à l'instruction `chaine = String.valueOf(tmp)`.

Ensuite, l'écriture de cette chaîne dans le fichier est réalisée par l'instruction `fW.write(chaine, 0, chaine.length())`. La méthode `write()` envoie dans le flux `fW` la chaîne spécifiée en premier paramètre. Les deuxième et troisième paramètres précisent respectivement à partir de quel caractère (`0`) commence l'écriture dans le fichier et combien de caractères (`chaine.length()`) sont écrits. Pour notre exemple, l'intégralité de la `chaine` est écrite dans le fichier.

Pour finir, la méthode `newLine()` envoie dans le flux `fW` un caractère permettant de passer à la ligne suivante du fichier.

- **La lecture** dans un fichier est décrite par la méthode :

```
public String lire() throws IOException {
  String chaine = fR.readLine();
  return chaine;
}
```

Copyright © 2016 Eyrolles.

L'opération de lecture est réalisée par la méthode `readLine()`, qui envoie tout d'abord la ligne lue sur le flux `fR` puis passe automatiquement à la ligne suivante dans le fichier. La chaîne de caractères `chaine` récupère alors la suite des caractères lus. Pour finir, la chaîne est retournée à la méthode appelante.

Fermeture du flux

Une fois que tous les traitements ont été réalisés, le flux peut être naturellement fermé grâce à la méthode :

```
public void fermer() throws IOException {
  if (mode == 'R' || mode == 'L') fR.close();
  else  if (mode == 'W' || mode == 'E') fW.close();
}
```

Suivant le mode d'ouverture spécifié par la variable d'instance `mode` (initialisée lors de l'exécution de la méthode `ouvrir()`), le flux `fR` ou `fW` est fermé grâce à la méthode `close()`.

Exemple : l'application GestionFichier

L'application suivante utilise les méthodes décrites ci-dessus pour créer et manipuler un fichier dont le nom est saisi au clavier :

```
import java.util.*;
public class GestionFichier {
 public static void main (String [] arg) throws IOException  {
  Scanner lectureClavier = new Scanner(System.in);
  Fichier f = new Fichier();
  System.out.print(" Entrer le nom du fichier : ");
  String nomFichier = lectureClavier.next();
  f.ouvrir(nomFichier, "Ecriture");
  for (int i = 0; i < 5; i++)  f.ecrire(i);
  f.fermer();

  f.ouvrir(nomFichier,"Lecture");
  String chaine ="";
  do {
    chaine = f.lire();
    if (chaine != null) System.out.println(chaine);
  } while (chaine != null);
  f.fermer();
 }
}
```

Copyright © 2016 Eyrolles.

L'instruction `f.ouvrir(nomFichier, "Ecriture")` ouvre le fichier `nomFichier` en écriture afin d'y écrire une suite de valeurs entières (`f.ecrire(i)`) comprises entre `0` et `4`. Le fichier est fermé (`f.fermer()`) après exécution de la boucle `for`.

Pour vérifier que les opérations d'écriture se sont bien déroulées, le fichier est ouvert en lecture (`f.ouvrir(nomFichier,"Lecture")`) et, grâce à une boucle `do...while`, chaque ligne du fichier est lue par `f.lire()` et mémorisée dans une variable `chaine` afin d'être affichée. La lecture de ce fichier prend fin lorsqu'une chaîne `null` est détectée (`while (chaine != null)`). Le fichier peut alors être fermé (`f.fermer()`).

L'exécution de cette application a pour résultat à l'écran :

```
Entrer le nom du fichier : Valeurs.txt
0
1
2
3
4
```

Remarque

Le fichier `Valeurs.txt` est créé dans le même répertoire que celui où se trouve l'application `GestionFichier.class`. Comme il s'agit d'un fichier texte, il peut être ouvert par n'importe quel éditeur de texte (Bloc-notes sous Windows, vi sous Unix ou encore TextEdit sous Mac OS). C'est là un des intérêts des fichiers textes.

Observez, cependant, que les données manipulées par un programme ne se résument généralement pas à de simples valeurs entières. Le plus souvent, une application travaille avec des objets complexes, mêlant plusieurs types de données. C'est pourquoi il est intéressant de pouvoir archiver, non pas la suite des données relatives à un objet, ligne par ligne, mais l'objet lui-même en tant que tel. Cette technique est examinée à la section suivante.

Les fichiers d'objets

Le langage Java propose des outils permettant le stockage ainsi que la lecture d'objets dans un fichier. Ces outils font appel à des mécanismes appelés mécanismes de sérialisation. Ils utilisent des flux spécifiques, définis par les classes `ObjectOutputstream` et `ObjectInputstream`, du package `java.io`.

La sérialisation des objets

Un objet est sérialisé afin de pouvoir être transporté sur un flux de fichier, entrant ou sortant. Grâce à cette technique, un objet peut être directement stocké dans un fichier (écriture) ou reconstruit à l'identique en mémoire vive par lecture du fichier.

Les mécanismes de sérialisation-désérialisation sont fournis par l'intermédiaire des classes `ObjectOutputstream` et `ObjectInputstream`, grâce aux méthodes `writeObject()`

Copyright © 2016 Eyrolles.

(sérialisation) et `readObject()` (désérialisation). Ces outils sont applicables à tous les objets prédéfinis du langage Java, tels que les `String`, les listes (`ArrayList`) ou encore les dictionnaires (`HashMap`).

Lorsque vous souhaitez sérialiser un objet dont le type est défini par le programmeur, il est nécessaire de rendre cet objet sérialisable. Pour cela, il suffit d'indiquer au compilateur que la classe autorise la sérialisation, en utilisant la syntaxe suivante :

```
public class Exemple implements Serializable {
// Données et méthodes
}
```

De cette façon, tous les objets déclarés de type `Exemple` peuvent être lus ou écrits dans des fichiers d'objets.

Remarque

L'objectif de la sérialisation est de placer, dans un flux, toutes les informations relatives à un objet. Par conséquent, seules les variables d'instance sont prises en compte lors d'une sérialisation.

Question

Les variables de classe peuvent-elles être sérialisées ?

Réponse

Les variables de classes (définies en *static*) ne peuvent être sérialisées. En effet, une variable de classe est commune à tous les objets de l'application et non pas spécifique à un seul objet. Elles ne peuvent donc être placées dans un flux relatif à un seul objet.

Exemple : archiver une classe d'étudiants

Pour bien comprendre comment utiliser ces outils d'archivage d'objets, modifions l'application `GestionCursus` de sorte qu'elle puisse lire et stocker automatiquement l'ensemble des données du dictionnaire `listeClassée` dans un fichier, portant le nom de `Cursus.dat`.

Nous devons tout d'abord rendre sérialisable les objets que nous souhaitons sauvegarder. C'est pourquoi il convient de modifier les en-têtes des classes `Etudiant` et `Cursus` de la façon suivante :

```
public class Etudiant implements Serializable {
// voir la section "Les dictionnaires"
}
public class Cursus implements Serializable {
// voir la section "Les dictionnaires"
}
```

À défaut, vous obtenez une erreur d'exécution du type `NotSerializableException`, indiquant que l'objet de type `Etudiant` ou `Cursus` ne peut être sérialisé.

Copyright © 2016 Eyrolles.

Ensuite, les opérations d'archivage d'objets utilisent les mêmes concepts que ceux décrits à la section précédente, à savoir ouverture du fichier, puis lecture ou écriture des objets et, pour finir, fermeture du fichier. C'est pourquoi nous allons modifier la classe `Fichier` pour y manipuler, non plus des fichiers textes, mais des fichiers d'objets.

```
import java.io.*;
public class FichierEtudiant {
 private ObjectOutputstream ofW;
 private ObjectInputstream ofR;
 private String nomDuFichier = "Cursus.dat";
 private char mode;
}
```

Les données de la classe `FichierEtudiant` sont deux objets représentant des flux d'écriture (`ofW`), de lecture (`ofR`) d'objets, ainsi qu'un caractère (`mode`) représentant le type d'ouverture du fichier et une chaîne de caractères (`nomDuFichier`), où se trouve mémorisé le nom de fichier de sauvegarde des données (`Cursus.dat`).

Ouverture du flux (entrant ou sortant)

```
public void ouvrir(String s) throws IOException   {
 mode = (s.toUpperCase()).charAt(0);
 if (mode == 'R' || mode == 'L')
   ofR  = new ObjectInputstream(new FileInputstream(nomDuFichier));
 else  if (mode == 'W' || mode == 'E')
   ofW  = new ObjectOutputstream(new FileOutputstream(nomDuFichier));
}
```

L'ouverture du fichier `Cursus.dat` en lecture est réalisée grâce aux constructeurs des classes `FileInputstream` et `ObjectInputstream`, alors que l'ouverture en écriture est effectuée par les constructeurs `ObjectOutputstream()` et `FileOutputstream()`. En résultat, les flux `ofW` et `ofR` contiennent les adresses de début de fichier.

Traitement du fichier

L'objectif est d'archiver l'ensemble des données relatives à une classe d'étudiants. La méthode `ecrire()` prend en paramètre un objet `tmp` de type `Cursus`, de sorte que l'information lui soit transmise depuis l'application `GestionCursus`. L'objet transmis est alors archivé grâce à la méthode `writeObject(tmp)`.

```
public void ecrire(Cursus tmp) throws IOException {
 if (tmp != null)  ofW.writeObject(tmp);
}
```

Inversement, la méthode `lire()` lit l'objet stocké dans le fichier `Cursus.dat` et le transmet en retour à l'application `GestionCursus` sous forme d'objet de type `Cursus`. L'en-tête de

Copyright © 2016 Eyrolles.

la méthode a pour type le type `Cursus`. L'objet retourné est lu grâce à la méthode `readObject()`.

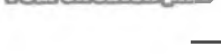
```
public Cursus lire() throws IOException, ClassNotFoundException  {
 Cursus tmp = (Cursus) ofR.readObject();
 return tmp;
}
```

Observons que :

- La méthode `lire()` traite obligatoirement un nouveau type d'exception : `ClassNotFoundException`. En effet, la méthode `readObject()` transmet ce type d'exception lorsque le fichier lu ne contient pas d'objet mais tout autre chose.

Pour plus de précisions sur la gestion des exceptions, voir la section « Gérer les exceptions », en fin de chapitre.

- La méthode `readObject()` lit sur le flux un objet, quel que soit son type. Il est donc nécessaire de spécifier, par l'intermédiaire d'un cast, le format de l'objet lu. Pour notre exemple, l'objet lu est transmis à l'objet `tmp` par l'intermédiaire d'un cast `(Cursus)`, qui réalise la transformation de l'objet au bon format.



Sur le mécanisme du cast, voir au chapitre 1, « Stocker une information », la section « La transformation de types ».

Fermeture du flux

La fermeture d'un flux est réalisée par la méthode `close()`, de la même façon qu'un flux de fichier texte.

```
public void fermer() throws IOException {
  if (mode == 'R' || mode == 'L') ofR.close();
  else  if (mode == 'W' || mode == 'E') ofW.close();
}
```

Exemple : l'application GestionCursus

L'application `GestionCursus` a pour contrainte de réaliser les actions suivantes :

- Une lecture automatique du fichier `Cursus.dat` dès l'ouverture du programme afin d'initialiser l'objet `C` (type `Cursus`) à la liste d'étudiants saisie lors d'une précédente exécution.
- Une sauvegarde automatique dans le fichier `Cursus.dat` lorsque l'utilisateur choisit de sortir du programme.

Ces deux contraintes sont réalisées par l'application suivante :

Copyright © 2016 Eyrolles.

```
import java.io.*;
import java.util.*;
public class GestionCursus  {
 public static void main (String [] argument)
            throws IOException, ClassNotFoundException {
  byte choix = 0 ;
  Scanner lectureClavier = new Scanner(System.in);
  Cursus C = new Cursus();
  FichierEtudiant F = new FichierEtudiant();
  F.ouvrir("Lecture");
  C = F.lire();
  F.fermer();
  String prénom, nom;
  do {
    System.out.println("1. Ajoute un etudiant");
    System.out.println("2. Supprime un etudiant");
    System.out.println("3. Affiche la liste des eleves");
    System.out.println("4. Affiche un etudiant");
    System.out.println("5. Sortir");
    System.out.println();
    System.out.print("Votre choix : ");
    choix  = lectureClavier.nextByte();
    switch (choix) {
      // pour les options 1, 2, 3, 4 voir
      // Exemple : Créer un dictionnaire d'étudiants
      case 1 : // Ajoute un etudiant
      case 2 : // Supprime un etudiant
      case 3 : // Affiche les etudiants
      case 4 : // Affiche un etudiant
      case 5 :
        System.out.println("Sauvegarde des donnees dans Cursus.dat");
        F.ouvrir("Ecriture");
        F.ecrire(C);
        F.fermer();
        System.exit(0) ;
      break;
      default :  System.out.println("Cette option n'existe pas ");
    }
  } while (choix != 5);
 }
} // Fin de la classe GestionCursus
```

Copyright © 2016 Eyrolles.

L'exécution de cette application montre qu'une difficulté subsiste. En effet, lors de la toute première exécution du programme, l'interpréteur affiche le message suivant :

```
java.io.FileNotFoundException:
Cursus.dat (Le fichier spécifié est introuvable)
  at java.io.FileInputstream.open(Native Method)
  at java.io.FileInputstream.<init>(FileInputstream.java:68)
  at FichierEtudiant.ouvrir(FichierEtudiant.java:14)
  at GestionCursus.main(Compiled Code)
```

L'erreur `FileNotFoundException` est transmise à la méthode `main()` via la méthode `FileInputstream.open()` grâce à la clause `throws IOException`.

En effet, le fichier `Cursus.dat` n'existe pas encore, puisque c'est la première fois que le programme est exécuté. L'option `5` n'a pu être exécutée, et aucune sauvegarde n'a donc été réalisée. Tant que le programme ne peut être exécuté dans son intégralité, aucun fichier de sauvegarde ne peut être créé.

Pour contourner cet obstacle, la solution consiste à empêcher les erreurs de remonter d'une méthode à l'autre grâce à la clause `throws`, tout en gérant de façon explicite chaque erreur qui pourrait survenir. Cette solution est examinée à la section suivante.

Gérer les exceptions

Une exception est générée par l'interpréteur Java lorsqu'un événement anormal ou inattendu survient. Les exceptions sont définies par les classes :

- `java.lang.Error`, pour des erreurs graves, qui conduisent à l'arrêt du programme ;
- `java.lang.Exception`, pour des événements qui peuvent être traités sans provoquer l'arrêt du programme.

Lorsqu'une exception est lancée, elle se propage et peut être capturée ou non par le programme.

Lorsque la clause `throws` est utilisée, l'exception se propage jusqu'à ce que le programme se termine. L'interpréteur retourne alors en résultat la liste des méthodes traversées par l'exception. L'erreur n'est pas capturée par le programme.

Pour capturer une exception, le langage Java propose des outils de capture d'erreurs afin de les traiter directement à l'intérieur des méthodes susceptibles de les détecter. Cette capture est réalisée par l'intermédiaire des instructions `try...catch`. Ces outils de capture d'erreurs ne sont pas spécifiques à la gestion des flux. Ils peuvent également être utilisés pour traiter des erreurs dues à la gestion des événements associés à une application graphique.

Pour connaître d'autres types d'exception , voir la section « De l'AWT à Swing », du chapitre 11.

Copyright © 2016 Eyrolles.

La méthode ouvrir()

Examinons le mécanisme des instructions `try...catch` sur la méthode `ouvrir()`, proposée à la section précédente. Comme observé précédemment, cette méthode pose problème, puisqu'elle propage l'erreur `FileNotFoundException` lors de la toute première exécution de l'application `GestionCursus`. Pour éviter cette propagation, l'idée est de placer le couple d'instructions `try...catch` de la façon suivante :

```
public boolean ouvrir(String s) {
  try {
    mode = (s.toUpperCase()).charAt(0);
    if (mode == 'R' || mode == 'L')
      ofR = new ObjectInputstream(new FileInputstream(nomDuFichier));
    else   if (mode == 'W' || mode == 'E')
    ofW = new ObjectOutputstream(new FileOutputstream(nomDuFichier));
    return true;
  }
  catch (IOException e)  {
    return false;
  }
}
```

La méthode s'exécute alors de la façon suivante :

Les instructions qui composent le bloc `try` (en français `essayer`) sont exécutées.

- Si aucune erreur n'est transmise par les différents constructeurs qui réalisent l'ouverture du fichier, le programme sort de la méthode `ouvrir()` en retournant un booléen de valeur égale à `true`.
- Si une erreur est propagée par l'un des constructeurs, les instructions placées dans le bloc `catch` (capture) sont exécutées, à condition que l'erreur détectée soit du même type que celui placé entre parenthèses derrière le terme `catch`. L'erreur `FileNotFoundException` étant du type `IOException`, le programme sort de la méthode `ouvrir()`, en retournant un booléen de valeur égale à `false`. Aucun fichier n'est donc ouvert.

Puisque la méthode `ouvrir()` capture et traite elle-même les erreurs éventuelles, la présence de la clause `throws` devient inutile, et elle n'apparaît plus dans l'en-tête de la méthode.

Grâce au traitement des erreurs en interne, les instructions relatives à l'ouverture du fichier en lecture dans l'application `GestionCursus` peuvent être modifiées de la façon suivante :

```
FichierEtudiant F = new FichierEtudiant();
if (F.ouvrir("L")) {
  C = F.lire();
  F.fermer();
}
```

Copyright © 2016 Eyrolles.

Le fichier est ouvert grâce à l'instruction `F.ouvrir()`. Si le résultat de la méthode vaut `true`, cela signifie que le fichier `Cursus.dat` existe et est ouvert. Les instructions de lecture du fichier situées dans le bloc `if` peuvent être exécutées. À l'inverse, si le résultat vaut `false`, aucune instruction n'est exécutée. Le menu permettant la saisie de nouveaux étudiants peut alors être affiché.

La méthode lire()

La méthode `lire()` est susceptible de lever plusieurs types d'exception, via la méthode `readObject()`. En effet, cette dernière est susceptible de détecter des erreurs du type `IOException` ou `ClassNotFoundException`.

La capture de ces exceptions est réalisée en définissant autant de blocs `catch` qu'il y a d'erreurs détectées. La méthode `lire()` traite ces erreurs de la façon suivante :

```
public Cursus lire() {
  try {
    Cursus tmp = (Cursus) ofR.readObject();
    return tmp;
  }
  catch (IOException e) {
    System.out.println(nomDuFichier + " : Erreur  de lecture ");
  }
  catch (ClassNotFoundException e) {
   System.out.println(nomDuFichier + " n'est pas du bon format ");
  }
  return null;
}
```

Remarque Il est possible de définir deux blocs `catch` successifs, paramétrés en fonction des types d'erreurs susceptibles d'être détectés. Le programme réagit alors différemment suivant l'erreur capturée.

Copyright © 2016 Eyrolles.

Résumé

La programmation **dynamique** permet la gestion d'un nombre indéterminé d'objets, en réservant des espaces mémoire au fur et à mesure des besoins de l'utilisateur.

Pour ce faire, le langage Java propose différents outils, tels que les objets de type `ArrayList` ou encore de type `HashMap`.

Les objets de type `ArrayList` autorisent la création d'une liste, par ajout de données au fur et à mesure des besoins de l'utilisateur. Les données sont, en général, enregistrées dans leur ordre d'arrivée. Un indice géré par l'interpréteur permet de retrouver l'information.

Pour utiliser une liste, il est nécessaire de la déclarer de la façon suivante :

```
ArrayList liste = new ArrayList() ;
```

Pour **ajouter un objet à la liste**, il suffit d'écrire `liste.add(objet)`. Il n'est pas possible d'ajouter une valeur autre qu'un objet (telle que les variables de type `int`, par exemple). Lorsque l'objet est inséré dans la liste, la taille de cette dernière est augmentée de un. La méthode `size()` calcule le nombre d'éléments dans la liste, et la méthode `get(indice)` permet de retrouver l'objet stocké à l'`indice` spécifié en paramètre.

La classe `ArrayList` étant définie dans le package `java.util`, il convient de placer l'instruction `import.java.util.*;` en tête du fichier. En effet, si cette instruction fait défaut, le compilateur détecte une erreur du type `Class ArrayList not found`.

La recherche d'éléments complexes dans une liste est plus rapide lorsque les données sont organisées, non plus par rapport à un indice, mais par rapport à une clé explicite. Les objets de type `HashMap` proposent ce type d'organisation des données. Pour cela, il suffit de déclarer une liste comme :

```
HashtMap liste = new Hashmap () ;
```

Les méthodes `put(cle, objet)` et `get(cle)` permettent respectivement de placer dans la liste (**dictionnaire**) l'association `cle-objet` et de retrouver l'objet associé à la `cle` spécifiée en paramètre.

Les types génériques permettent de forcer les listes ou les dictionnaires à n'enregistrer que des données du type spécifié entre les deux signes < et >. Ainsi la déclaration :

```
ArrayList<String> liste = new ArrayList<String>() ;
```

crée une liste ne pouvant contenir que des objets de type `String`.

Copyright © 2016 Eyrolles.

Grâce aux streams et expressions lambda, il devient plus facile de traiter rapidement les collections de données, en utilisant des instructions sous forme condensée :

```
liste.stream().filter(e -> e.quelNom().startsWith("A"))
              .forEach(e -> e.afficheUnEtudiant());
```

Pour éviter que les données stockées en mémoire vive de l'ordinateur ne se perdent à l'arrêt de l'application, il est nécessaire de les archiver sous forme de fichiers sur le disque dur. Pour cela, le langage Java utilise le concept de flux de fichier (en anglais `stream`), qui est, en quelque sorte, la concrétisation informatique du courant électrique passant de la mémoire vive au disque dur de l'ordinateur.

Il existe différents types de **flux de fichiers** :

- D'une part, les flux **entrant**, pour lire les données sur le disque dur et les placer en mémoire vive, et les flux **sortant**, qui écrivent les données de la mémoire vive sur le disque dur.
- D'autre part, les fichiers de type texte (`BufferedWriter`, `BufferedReader`), qui ne font que manipuler des données de type `String`, et les fichiers d'objets (`ObjectOutputstream`, `ObjectInputstream`), qui manipulent tout type d'objet.

D'une façon générale, les traitements sur fichiers se déroulent en trois temps : **ouverture** du flux, **traitement** des données parcourant le flux, puis **fermeture** du flux. Lorsqu'un fichier est ouvert en écriture :

- Si le fichier n'existe pas, et :
  - Si le chemin d'accès à ce fichier dans l'arborescence du disque est valide, alors le fichier est créé.
  - Si le chemin d'accès n'est pas valide, alors le fichier n'est pas créé et une erreur du type `FileNotFoundException` est détectée.
- Si le fichier existe, il est ouvert, et son contenu est **totalement effacé**.

Lorsqu'une erreur est détectée par les méthodes associées au flux, le couple d'instructions `try...catch` permet la **capture** de l'exception afin de lui associer un traitement spécifique.

Copyright © 2016 Eyrolles.

Exercices

Comprendre les listes

Exercice 10.1 L'objectif est de stocker les notes d'un étudiant sous la forme d'une liste.

a. Définissez un objet `note` de type `ArrayList` comme variable d'instance de la classe `Etudiant`. La liste `note` ne peut contenir que des valeurs de type `Double`.

b. Modifiez le constructeur de la classe `Etudiant` afin de saisir les notes et de les placer dans la liste.

Prenez garde que seul un objet peut être stocké dans une liste. Une note, étant de type `double` (type simple), ne peut pas être directement placée dans la liste. Il est nécessaire de la transformer en objet de type `Double`. L'appel au constructeur de la classe `Double` permet cette transformation.

Remarque L'instruction `new Double(lectureClavier.nextDouble())` permet la transformation directe d'une valeur `double` saisie au clavier en un objet de type `Double`.

c. La méthode `calculMoyenne()` doit calculer la moyenne des notes à partir des notes saisies dans le constructeur. Le programme doit, par conséquent, parcourir l'ensemble de la liste `note` afin d'en calculer la somme. Vous utiliserez pour cela, la nouvelle syntaxe de la boucle `for`.

d. Dans la méthode `afficheUnEtudiant()`, modifiez l'affichage des notes en parcourant, non plus le tableau, mais la liste `note`, et ce en utilisant la nouvelle syntaxe de la boucle `for`.

Exercice 10.2 Reprendre l'application `FaireDesFormesGeometriques` et les classes `Forme`, `Triangle` et `Rectangle` développées au cours des exercices 8.6 à 8.8 du chapitre 8, « Les principes du concept d'objet », ainsi que la classe `Cercle`, décrite au cours de ce même chapitre.

Extension Web Pour vous faciliter la tâche, vous trouverez dans le répertoire `Source/Exercices/Chapitre10/SupportPourRealiserLesExercices/Forme` sur l'extension Web de l'ouvrage, tous les fichiers nécessaires à la réalisation de cette application.

L'objectif est de créer une liste de `Forme` à l'aide de la classe `ListeDeFormes`. Cette dernière est composée :

a. D'une propriété `listeFormes` de type `ArrayList`. En utilisant les types génériques, faites en sorte que la liste ne puisse contenir que des objets de type `Forme`.

b. D'un constructeur initialisant la propriété `listeFormes`.

Copyright © 2016 Eyrolles.

c. D'une méthode `ajouterUneForme()` qui permet l'ajout d'un cercle, d'un rectangle ou d'un triangle dans la liste suivant la valeur fournie en paramètre de la méthode. Par exemple, `ajouteUneForme('C')` a pour résultat de créer et d'insérer un cercle dans la liste `listeFormes`.

d. D'une méthode `afficherLesFormes()` qui affiche les objets de la `listeFormes` en parcourant à l'aide d'une boucle `for` l'ensemble de la liste. Est-il besoin de préciser le type de l'objet à afficher (`Cercle`, `Triangle` ou `Rectangle`) ? Pourquoi ?

e. Créer l'application `FaireDesListesDeFormes` qui réalise les opérations proposées par le menu suivant :

```
1. Ajouter un cercle
2. Ajouter un triangle
3. Ajouter un rectangle
4. Afficher la liste
5. Pour sortir
Votre choix :
```

Exercice 10.3 L'objectif est d'écrire une méthode qui affiche la liste de tous les étudiants portant le même prénom, en utilisant les streams et expressions lambda. La méthode a pour nom `memePrenom()`.

a. Dans la classe `Etudiant` écrire la méthode `quelPrenom()` qui retourne le prénom de l'étudiant.

b. Dans la classe `Cursus` écrire la méthode `memePrenom()` qui affiche la liste des étudiants portant le prénom passé en paramètre de la méthode. Pour cela, vous pouvez vous inspirer du code fourni en exemple du cours.

c. Dans l'application `GestionCursus` ajouter une quatrième option au menu qui permet de saisir au clavier le prénom recherché et d'appeler la méthode `memePrenom()` sur la liste des étudiants.

Exercice 10.4 L'objectif est d'écrire une méthode qui affiche le major de la promotion en utilisant les streams et expressions lambda. La méthode a pour nom `rechercheLeMajor()`.

a. Dans la classe `Etudiant` écrire la méthode `quelleMoyenne()` qui retourne la moyenne de l'étudiant.

b. Dans la classe `Cursus` écrire la méthode `rechercheLeMajor()` qui retourne l'étudiant qui a la meilleure moyenne de la promotion. Pour information, l'expression lambda qui calcule la plus grande valeur d'une liste de nombres s'écrit comme suit :

```
liste.stream().max((a,b) -> (a.quelleValeur() -.
b.quelleValeur()))
```

L'expression `get()` appliquée ensuite à l'expression `max()` permet de récupérer la plus grande valeur trouvée par `max()`.

c. Dans l'application `GestionCursus` ajouter une cinquième option au menu qui permet d'afficher l'étudiant major de la promotion.

Copyright © 2016 Eyrolles.

Comprendre les dictionnaires

Exercice 10.5 L'objectif est d'écrire une méthode `modifieUnEtudiant()` qui modifie les notes d'un étudiant stocké dans un dictionnaire. Cette méthode fonctionne dans l'ensemble comme la méthode `ajouteUnEtudiant()` (voir la section « Les dictionnaires » de ce chapitre).

a. Cependant, la méthode doit connaître les `nom` et `prenom` de l'étudiant à modifier. Ces données lui sont transmises par paramètre.

b. Ensuite, connaissant les `nom` et `prenom`, le programme calcule la `cle` et vérifie si l'étudiant existe dans la liste.

c. S'il existe, la modification consiste à lui donner de nouvelles `notes`. Pour cela, l'idée est d'écrire un deuxième constructeur `Etudiant()`, dont les paramètres sont les `nom` et `prenom` de l'étudiant. Le corps du constructeur ne fait ensuite que stocker dans les variables d'instance appropriées les `nom` et `prenom` passés en paramètres, sans avoir à les ressaisir, puis saisir les nouvelles `notes` et enfin calculer la `moyenne`.

d. Modifiez l'application `GestionCursus` de façon à intégrer au menu cette nouvelle option.

Exercice 10.6 Reprendre l'application `Bibliotheque` et la classe `Livre` développées au cours des exercices 8.1 à 8.5 du chapitre 8, « Les principes du concept d'objet ».

Extension Web Pour vous faciliter la tâche, vous trouverez dans le répertoire `Source/Exercices/Chapitre10/SupportPourRealiserLesExercices/Livre` sur l'extension Web de l'ouvrage, tous les fichiers nécessaires à la réalisation de cette application.

L'objectif est de créer un dictionnaire composé de `Livre` à l'aide de la classe `ListeDeLivres`. Cette dernière est composée :

a. D'une propriété `liste` de type `HashMap`. En utilisant les types génériques, faites en sorte que le dictionnaire ne puisse contenir que des objets de type `Livre`.

b. D'un constructeur initialisant la propriété `liste`.

c. D'une méthode `ajouterUnLivre()` qui permet l'ajout d'un livre dans le dictionnaire. La clé d'association est calculée grâce à la méthode `getCode()` définie dans la classe `Livre`.

d. D'une méthode `rechercherUnLivre()` qui affiche le livre dont le nom, le prénom de l'auteur, la catégorie et le numéro ISBN sont fournis en paramètre de la méthode.

Remarque Le calcul de la clé s'effectue en créant un livre temporaire grâce au constructeur de la classe `Livre`. Le calcul du code d'un livre ne nécessite pas de connaître son titre, vous pouvez passer en paramètre du constructeur la valeur `null`, en lieu et place du paramètre représentant le titre du livre.

Copyright © 2016 Eyrolles.

Si le livre est retrouvé dans la `liste`, la méthode `rechercherUnLivre()` affiche les informations le concernant.

e. D'une méthode `supprimerUnLivre()` qui supprime le livre dont le nom, le prénom de l'auteur, la catégorie et le numéro ISBN sont fournis en paramètre de la méthode.
   - Le calcul de la clé s'effectue selon la méthode décrite au point d.
   - Si le livre est retrouvé dans la `liste`, la méthode `supprimerUnLivre()` supprime le livre de la `liste`.

f. D'une méthode `afficherLesLivres()` qui affiche les objets de la `liste` en parcourant, à l'aide d'une boucle `for`, l'ensemble du dictionnaire.

g. Créer l'application `Bibliotheque` qui réalise les opérations proposées par le menu suivant :

```
1. Ajouter un livre
2. Supprimer un livre
3. Afficher la liste des livres
4. Afficher un livre
5. Sortir
```

Remarque Les options `2` et `4` demandent la saisie des nom et prénom de l'auteur, de la catégorie du livre ainsi que du numéro ISBN pour obtenir la clé d'association et effectuer les recherches dans la liste. Vous pouvez simplifier la tâche de l'utilisateur en lui proposant de ne saisir que les 2 derniers chiffres du numéro ISBN.

Créer des fichiers textes

Exercice 10.7 L'objectif est de créer un fichier `Formes.txt` contenant les données relatives à chaque forme géométrique enregistrée dans la liste `listeFormes` créée au cours de l'exercice 10.2.

Le fichier possède autant de lignes qu'il y a d'objets placés dans la liste. Chaque forme est représentée par une chaîne de caractères spécifique :

- Un cercle est représenté par `C;couleur;x;y;rayon`.
- Un rectangle par `R;couleur;x;y;largeur;hauteur`.
- Un triangle par `T;couleur;x;y;x1;y1;x2;y2;`.

a. Examinez la méthode `getInfos()` de la classe `Forme` ci-après et donnez la structure de la chaîne retournée par la méthode.

```
public String getInfos() {
return couleur+";"+x+";"+y;
}
```

- Pour chacune des classes fille (`Triangle`, `Cercle`, ...), écrire une méthode `getInfos()` en faisant appel à la méthode `getInfos()` de la classe mère.
- Faites en sorte que chacune des chaînes résultantes corresponde au format demandé.

Copyright © 2016 Eyrolles.

b. Pour enregistrer dans le fichier `Formes.txt`, les données relatives à chaque forme géométrique, la technique consiste à définir dans la classe `ListeDeFormes`, une méthode `enregistrerLesFormes(Fichier f)` qui :
   - parcourt la liste `listeFormes` et récupère grâce à la méthode la méthode `getInfos()` les informations sous la forme d'une chaîne de caractères ;
   - enregistre la chaîne en utilisant la méthode `ecrire()` de la classe `Fichier` proposée en section « Les fichiers textes ».

c. L'enregistrement est effectif, lorsque la méthode `enregistrerLesFormes()` est appelée depuis l'application `FaireDesListesDeFormes`. Pour cela :
   - créez un objet `F` de type `Fichier` ;
   - appelez la méthode `enregistrerLesFormes()` lorsque l'utilisateur choisit de sortir de l'application, en l'appliquant sur l'objet `LdF` et, en passant en paramètre l'objet `F`.

d. Pour lire les informations enregistrées dans le fichier `Formes.txt`, il convient de mettre en place un mécanisme de découpage de chaque ligne lue, de façon à récupérer les informations comme la position en `x`, `y`, le type de la forme, ...

   Pour cela, vous devez modifier le méthode `lire()` de la classe `Fichier` comme suit :
   - La méthode lit une ligne du fichier à la fois et l'enregistre dans la chaîne de caractère `ligne`.
   - La chaîne `ligne` est découpée en utilisant un objet `StringTokenizer` comme suit :

```
import java.util.*;
StringTokenizer st = new StringTokenizer(ligne,separateur);
int i=0;
String mot[] = new String[st.countTokens()];
while (st.hasMoreTokens())  {
  mot[i] = st.nextToken();
  i++;
}
```

Remarque La classe `StringTokenizer` est une classe du paquetage `java.util`, dans laquelle est défini un ensemble de méthodes permettant l'extraction de mots dans une chaîne de caractères. Chaque mot doit être séparé par un caractère donné (ici, vous prendrez le séparateur `";"`).

   L'exécution de ces instructions a pour résultat de placer dans `mot[0]` le caractère correspondant au cercle (`C`), au rectangle (`R`) ou au triangle (`T`), dans `mot[1]` la couleur, dans `mot[2]` la position en `x`, etc.

   La méthode `lire()` retourne en résultat le tableau `mot`.

e. Pour lire l'ensemble du fichier, définissez dans la classe `ListeDeFormes`, la méthode `lireLesFormes(Fichier f)`. Cette méthode :
   - Parcourt le fichier jusqu'à obtention d'une ligne `null`.
   - Récupère, pour chaque ligne lue grâce à la méthode `lire()`, un tableau dont le premier élément est testé et s'il contient le caractère `'C'`, la méthode crée un cercle à l'aide du constructeur `Cercle()` en passant en paramètre les valeurs contenues dans la suite du tableau.
   - Construit de la même façon un rectangle si le caractère est un `'R'`, un triangle si c'est un `'T'`.
   - Les objets crées sont ajoutés à la liste `listeFormes`.

Copyright © 2016 Eyrolles.

f. Dans l'application `FaireDesListesDeFormes`, tester dès le début de l'application, l'ouverture en lecture du fichier `Formes.txt`. Si l'ouverture s'exécute correctement, lire le fichier et stocker les objets lus dans l'objet `LdF` à l'aide de la méthode `lireLesFormes`.

Remarque Pour vérifier que l'ouverture du fichier texte s'est bien déroulée, vous devez modifier la méthode `ouvrir()` de la classe `Fichier`, en vous inspirant de celle présentée en section « Gérer les exceptions – La méthode ouvrir() » de ce chapitre.

Créer des fichiers d'objets

Exercice 10.8 L'objectif est de créer un fichier `Bibliotheque.dat` contenant la liste des livres enregistrés dans la liste `liste` créée au cours de l'exercice 10.4.

a. Reprendre la classe `FichierEtudiant` construite en section « Les fichiers d'objets – Exemple : archiver une classe d'étudiants », nommer la `FichierDeLivres`. Modifiez cette dernière, de façon à :
   - nommer le fichier d'enregistrement des objets, `Bibliotheque.dat`.
   - lire ou enregistrer des objets de type `ListeDeLivres`.

b. Modifiez l'application `Bibliotheque` et créez un objet `F` de type `FichierDeLivres`.

c. Testez dès le début de l'application, l'ouverture en lecture du fichier `FichierDeLivres`. Si l'ouverture s'exécute correctement, lire le fichier et stocker les objets lus dans l'objet `LdL` à l'aide de la méthode `lire()` de la classe `FichierDeLivres`.

d. Pour sauvegarder les objets créés par l'utilisateur lorsque ce dernier choisit de sortir de l'application, ouvrir le fichier en écriture et enregistrer les informations grâce à la méthode `ecrire()` de la classe `FichierDeLivres`.

Gérer les erreurs

Exercice 10.9 L'objectif est de capturer toutes les erreurs (`IOException`) possibles dans la classe `FichierEtudiant` décrite au cours de ce chapitre.

a. Reprenez la classe `FichierEtudiant`, et gérez la détection des erreurs pour les méthode `fermer()` et `ecrire()`, en définissant des blocs `catch` et `try` appropriés.

b. Lorsque toutes les méthodes de la classe `FichierEtudiant` gèrent les exceptions, plus aucune clause `throws` ne doit apparaître sur l'en-tête des méthodes, y compris pour la méthode `main()` de l'application `GestionCursus`. Modifiez l'application `GestionCursus` en tenant compte de cette remarque.

Exercice 10.10 Reprendre la classe `EncodageParDefaut` donnée en exemple au chapitre 2, « Communiquer une information », à la section « Les caractères spéciaux ».

Copyright © 2016 Eyrolles.

Sachant que l'encodage d'une chaîne de caractères peut entraîner une erreur du type `UnsupportedEncodingException`, écrire un bloc `try` et un bloc `catch` réalisant la capture de cette exception.

a. Le bloc `try` est utilisé pour réaliser l'encodage de la chaîne `proverbe`.

b. Le bloc `catch` affiche un message indiquant que le codage par défaut n'est pas supporté par l'interpréteur Java.

Le projet : Gestion d'un compte bancaire

Les comptes sous forme de dictionnaire

La classe ListeCompte

En reprenant la classe `Cursus`, présentée au cours de ce chapitre, écrire la classe `ListeCompte` dont la donnée est une liste de type `HashMap`. La classe `ListeCompte` est composée des méthodes suivantes :

a. `ListeCompte()` qui fait appel au constructeur de la classe `HashMap`.

b. `ajouteUnCompte(String t)` qui permet la création d'un compte courant, joint ou d'épargne. Afin de faire appel au constructeur approprié (`Compte()` ou `CpteEpargne()`), faire passer en paramètre de la méthode `ajouteUnCompte()` une chaîne de caractères spécifiant le type du compte à créer. Par exemple, lorsque le paramètre de la méthode vaut `"E"`, un compte d'épargne est créé, alors que s'il vaut `"A"` (comme `Autre`), un compte ordinaire est créé.

   Lorsque le compte est créé, insérez-le dans le dictionnaire, en prenant comme clé d'association son numéro de compte.

c. `ajouteUneLigne()`, qui ajoute une ligne au compte dont le numéro est spécifié en paramètre de la méthode. Pour cela, faites appel à la méthode `créerLigne()` de la classe `Compte`.

d. Les méthodes `rechercheUnCompte()`, `supprimeUnCompte()` et `afficheLesComptes()` sont à écrire en s'inspirant des méthodes équivalentes de la classe `Cursus`.

L'application Projet

Dans l'application `Projet`, déclarer l'objet `C` comme étant du type `ListeCompte`. Puis :

a. Dans chaque option du menu, faire appel aux méthodes de la classe `ListeCompte`.

b. Lors de l'ajout d'un compte, ne pas omettre de spécifier en paramètre le type du compte (`"A"`, ou `"E"`).

c. Ajouter l'option de suppression d'un compte (option `5`) et l'affichage de la liste de tous les comptes (option `3`). Modifier l'affichage du menu et le `switch` de façon à tenir

Copyright © 2016 Eyrolles.

compte de ces nouvelles options. Notons qu'il n'est plus besoin de tester l'existence du compte avant de l'afficher ou de le supprimer, puisque ce sont les méthodes de la classe `ListeCompte` qui s'en chargent directement.

La sauvegarde des comptes bancaires

La classe FichierCompte

Pour sauvegarder les données saisies pour chaque compte, reprendre la classe `FichierEtudiant` décrite dans ce chapitre :

a. Modifier le nom de la classe par `FichierCompte`, et remplacer le nom du fichier de sauvegarde par `Compte.dat`.

b. Dans les méthodes `lire()` et `ecrire()`, remplacer l'objet lu ou écrit par un objet de type `ListeCompte`.

c. Ne pas oublier de rendre sérialisable l'ensemble des classes nécessaires à la construction de la liste des comptes.

L'application Projet

Modifier l'application, de façon à :

a. lire le fichier `Compte.dat` avant de proposer l'ajout, la suppression ou l'affichage des comptes ;

b. réaliser une sauvegarde automatique à la sortie du programme (option `6`).

La mise en place des dates dans les lignes comptables

Chaque ligne comptable est définie par un ensemble de données, dont la date de réalisation de l'opération. Pour l'instant, cette date est saisie sous forme d'un `String`, sans aucun contrôle sur le format réellement saisi (jour/mois/an). L'objectif est d'écrire une méthode qui vérifie si les valeurs saisies correspondent au format demandé.

Rechercher des méthodes dans les différents packages

Pour effectuer ce contrôle, le langage Java propose un certain nombre d'outils définis dans les packages du JDK. En particulier, il existe des outils qui transforment une chaîne de caractères en objet `Date`. Cette transformation est réalisée à partir d'un format défini par le programmeur.

Pour trouver ces différents outils, les deux solutions suivantes sont possibles :

a. Soit rechercher dans l'arborescence du JDK fourni sur l'extension Web de l'ouvrage (voir l'annexe « Guide d'installations », section « Installer la documentation en ligne ») tous les fichiers contenant le mot `Date` afin de déterminer les différents packages

Copyright © 2016 Eyrolles.

concernés par ce type d'information. Puis, pour tous les fichiers trouvés, examiner les différentes méthodes proposées, de façon à trouver celle susceptible de répondre à votre attente.

b. Soit se connecter sur Internet, par exemple à l'adresse *http://forum.java.sun.com*, de façon à y rechercher des exemples utilisant des objets de type `Date`.

Écrire la méthode contrôleDate()

L'algorithme permettant le contrôle du format de la date est le suivant :

a. Saisir une date comme une suite de caractères (`String`).

b. Traduire cette chaîne en objet `Date` grâce aux méthodes trouvées à l'étape précédente.

c. Capturer les erreurs propagées par cette méthode afin d'incrémenter un compteur d'essai de saisie de la date.

d. Répéter ces deux derniers points tant que la date n'est pas correctement traduite (l'objet `date` restant égal à `null`). Au bout de trois essais, la date est initialisée à la date courante du système de l'ordinateur.

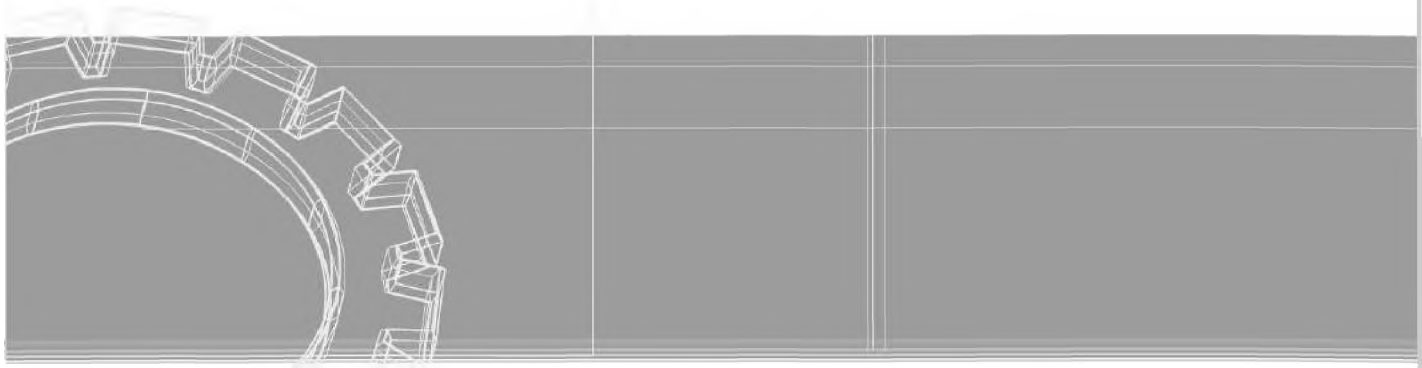
e. En sortie de boucle, la date correspond au format demandé. Elle peut être traduite en type `String`, pour être ensuite stockée dans la donnée `date`, de la classe `LigneComptable`.

Copyright © 2016 Eyrolles.

Copyright © 2016 Eyrolles.

Chapitre 11

Dessiner des objets



Le langage Java s'est surtout fait connaître en proposant pour Internet des outils de développement d'applications graphiques multi-plates-formes, c'est-à-dire fonctionnant sur des ordinateurs de tout type. Ces programmes sont exécutés à travers un navigateur Web de façon transparente pour l'internaute, que l'ordinateur utilisé soit un Mac, un PC ou une station Unix.

Ces applications utilisent des composants graphiques définis dans la bibliothèque graphique AWT (*Abstract Windowing Toolkit*). Dans ce chapitre, nous étudions d'abord, à la section « La bibliothèque AWT », comment utiliser les outils de ce package. Nous abordons ensuite, à la section « Les événements », la gestion des événements en analysant comment associer une action, ou un comportement, à un composant graphique. Pour finir, nous examinerons à la section « De l'AWT à Swing » comment fonctionnent les composants graphiques de la bibliothèque Swing.

La bibliothèque AWT

La bibliothèque AWT est un package du JDK (*Java Development Kit*), qui propose un ensemble d'outils de création d'applications graphiques, c'est-à-dire d'applications dont le mode de communication avec l'utilisateur s'établit à travers des éléments graphiques, tels que boutons, menus, fenêtres, etc.

Copyright © 2016 Eyrolles.

Remarque Notre objectif n'est pas de décrire l'intégralité de la bibliothèque AWT mais de présenter au lecteur un certain nombre d'exemples afin de lui donner une bonne vision de l'utilisation de ces outils, ainsi qu'une certaine méthodologie.

Pour cela, nous reprenons l'exemple du sapin de Noël décoré, décrit à la section « Les tableaux à deux dimensions » du chapitre 9, « Collectionner un nombre fixe d'objets ». Cette fois, le sapin n'est plus affiché à l'aide de simples caractères mais avec des composants graphiques utilisant les méthodes prédéfinies de la bibliothèque AWT.

Les fenêtres

L'affichage d'un outil graphique quel qu'il soit (bouton, menu, etc.) est toujours réalisé dans une **fenêtre**. Toute application graphique s'exécute à l'intérieur d'une zone délimitée, appelée fenêtre principale, dans laquelle sont placés barres d'outils, menus et zones de texte ou de dessin.

Cette fenêtre délimite le cadre d'exécution du programme, et tout élément se situant en dehors de la fenêtre fait partie d'une autre application. La fenêtre possède un bord et une barre de titre, dans laquelle se situent des boutons de fermeture et de mise en icône ou d'agrandissement, comme illustré à la figure 11-1. Elle peut être déplacée ou agrandie sans que le programmeur ait à gérer lui-même ces actions.

En langage Java, la fenêtre principale est définie grâce à la classe `Frame`. Observons le programme suivant, qui décrit comment définir et afficher une `Frame`.

Exemple : une fenêtre

```
import java.awt.*;
public class Fenetre {
 public  final static int HT = 300;
 public   final static int LG = 300;
 public static void main(String [] arg) {
  Frame F = new Frame();
  F.setTitle("Une fenetre!");  // met le titre
  F.setSize(LG, HT);           // taille de la fenêtre
  F.setBackground(Color.gray);
  F.setVisible(true);          // affiche la fenêtre
 }
}
```

Copyright © 2016 Eyrolles.

Nous constatons tout d'abord que la toute première instruction d'un programme qui utilise des objets graphiques est obligatoirement une instruction d'import du package de la bibliothèque AWT (`import java.awt.* ;`). En effet, comme pour les vecteurs et les dictionnaires, les outils de la bibliothèque graphique ne sont pas directement connus du compilateur.

Figure 11-1 *La fenêtre principale délimite le lieu d'exécution du programme. Elle est constituée d'une bordure et d'une barre de titre.*

Après avoir défini deux constantes, `HT` et `LG`, pour la hauteur et la largeur de la fenêtre, la fonction `main()` déclare et construit un objet `F` de type `Frame` (`Frame F = new Frame();`). Comme toute classe, la classe `Frame` propose un ensemble de méthodes qui permettent la transformation de ses caractéristiques, notamment les suivantes :

- `setTitle()`, qui place la chaîne de caractères spécifiée en paramètre dans la barre de titre de la fenêtre.
- `setSize()`, qui définit la hauteur et la largeur de la fenêtre.
- `setBackground()`, qui donne une couleur de fond à la fenêtre.

Cela fait, la fenêtre est définie en mémoire mais n'est pas encore affichée à l'écran. Pour réaliser cet affichage, la méthode `setVisible()`, définie par la classe `Frame`, est appliquée à l'objet `F`. La fenêtre `F` est visible lorsque le paramètre de la méthode `setVisible()` vaut `true`.

Pour connaître en détail l'ensemble des fonctionnalités de la classe `Frame`, reportez-vous au fichier `C:\jdk1.5\docs\api\java\awt\Frame.html`, après installation du JDK et de sa documentation.

Exemple : résultat de l'exécution

Lors de l'exécution de ce programme, la fenêtre ayant pour titre `Une fenetre !` apparaît à l'écran, comme illustré à la figure 11-1.

Copyright © 2016 Eyrolles.

Le dessin

Une fois affichée, la fenêtre n'est pas encore directement fonctionnelle, et il n'est pas possible d'y afficher un dessin ou d'y écrire un texte. Il n'est pas non plus possible de fermer la fenêtre en cliquant sur le bouton de fermeture situé dans la barre de titre.

En effet, l'affichage d'un dessin ne peut être réalisé que par l'intermédiaire d'un objet de type `Canvas`.

En outre, pour fermer la fenêtre d'un simple clic sur le bouton approprié, le programme doit être capable « d'entendre » les clics de la souris. Nous étudions ce concept à la section « Les événements », en fin de chapitre.

Exemple : dessiner un sapin de Noël

L'objectif de cet exemple est de réaliser l'affichage d'un sapin décoré en mode graphique. Fidèles à la méthode de travail qui consiste à découper un problème en plusieurs tâches indépendantes, nous allons réaliser l'affichage du sapin de Noël étape par étape.

Prenons pour hypothèse qu'un sapin est formé de triangles, disposés à l'écran de façon à ce que leur juxtaposition réalise une forme de sapin. Nous placerons ensuite la décoration du sapin en modifiant la couleur de certains triangles.

Nous devons concevoir dans un premier temps, un programme qui dessine un simple triangle de couleur verte.

Dessiner un triangle

Pour cela, nous définissons une classe `Dessin` qui hérite de la classe `Canvas` (`class Dessin extends Canvas`). De cette façon, un objet de type `Dessin` correspond à une zone d'affichage où il est possible de dessiner des formes géométriques (point, droite, rectangle, etc.). Examinons attentivement cette classe :

```
import java.awt.*;
public class Dessin extends Canvas {
 public Dessin()  {
  setBackground(Color.white);
  setForeground(Color.green);
  setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
 }
 public void paint (Graphics g)  {
  new Triangle(g);
 }
}
```

Copyright © 2016 Eyrolles.

La classe `Dessin` est composée des deux méthodes suivantes :

- Le constructeur `Dessin()`, qui initialise une partie des caractéristiques d'un objet `Canvas`, à savoir :
  - La couleur de fond. La méthode `setBackground(Color.white)` place la couleur blanche en fond de la zone de dessin (`Canvas`).
  - La couleur d'avant-plan. La méthode `setForeground(Color.green)` assigne la couleur verte aux formes géométriques dessinées dans la zone de dessin.
  - Le curseur. La méthode `setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR))` affiche un curseur en forme de croix lorsque le curseur de la souris se situe dans la zone de dessin.
- La méthode `paint()`, qui est une méthode prédéfinie de la classe `Canvas`. Cette méthode est appelée par l'interpréteur dès qu'il lui est nécessaire d'afficher un objet graphique. Elle est appelée lors de l'affichage de la fenêtre principale ou lorsque cette dernière réapparaît, après avoir été partiellement ou totalement cachée par une autre fenêtre.

La méthode `paint()` utilise en paramètre un objet `g` de type `Graphics` de façon à obtenir des informations sur le contexte graphique défini par l'application.

Remarque

Le contexte graphique est l'ensemble des informations utiles à l'affichage d'un objet. Par exemple, la couleur et la forme des caractères (fonte) font partie du contexte graphique.

Ainsi, lorsque l'interpréteur affiche une fenêtre, il transmet à la méthode `paint()`, par l'intermédiaire du paramètre `g`, toutes les caractéristiques de l'affichage. En particulier, il lui transmet sa couleur d'avant-plan, initialisée à `color.green` dans le constructeur `Dessin()`.

Pour finir, l'exécution de la méthode `paint()` réalise l'affichage du triangle grâce à l'appel du constructeur de la classe `Triangle` (`new Triangle(g)`), dont voici la description :

```
import java.awt.*;
public class Triangle {
 private int centreX = Fenetre.LG/2;
 private int centreY = Fenetre.HT/2;
 private int [] xPoints = {centreX, centreX + 10, centreX - 10};
 private int [] yPoints = {centreY - 10, centreY + 10, centreY +
10};
 int nPoints = 3;
 public Triangle(Graphics g) {
  g.fillPolygon(xPoints, yPoints, nPoints);
 }
}
```

Copyright © 2016 Eyrolles.

Les données de la classe `Triangle` correspondent à deux tableaux d'entiers définissant les sommets d'un triangle centré, comme illustré à la figure 11-2.

Figure 11-2 *Le triangle est construit à partir des sommets* `A`, `B` *et* `C`, *dont les coordonnées sont calculées par rapport à l'origine de la zone de dessin située en haut et à gauche.*

Le sommet `A` est défini par le couple de coordonnées `(xPoints[0], yPoints[0])`, le sommet `B` par `(xPoints[1], yPoints[1])` et `C` par `(xPoints[2], yPoints[2])`. Les tableaux définissent ainsi les sommets d'un polygone (triangle), centré par rapport à la fenêtre principale de l'application.

Remarque

Les coordonnées des sommets sont définies par rapport à l'origine de l'objet `Canvas`, laquelle est située par défaut dans le coin supérieur gauche de cet objet.

L'affichage du triangle est réalisé grâce à la méthode `fillPolygon(xPoints, yPoints, nPoints)`, qui remplit de couleur le polygone spécifié en paramètre. La couleur de remplissage est déterminée par l'intermédiaire de l'objet `g` sur lequel la méthode est appliquée.

L'application Fenetre

Sans modification de l'application `Fenetre`, telle que définie à la section précédente de ce chapitre (voir « Les fenêtres »), aucun triangle n'apparaît. En effet, avant d'exécuter le programme, nous devons ajouter à la fenêtre le composant `Dessin`, de sorte que l'application puisse associer l'objet de type `Canvas` à la `Frame F`. Pour cela, il suffit d'ajouter l'instruction `F.add(new Dessin())`, comme l'illustre l'extrait de programme suivant :

```
public class Fenetre {
  //…
  Frame F = new Frame();
  F.setTitle("Un triangle");
```

Copyright © 2016 Eyrolles.

```
    //…
    F.add(new Dessin());
    F.setVisible(true);          // affiche la fenêtre
  }
}
```

La méthode `add()` ajoute un composant graphique à la fenêtre principale. Ce composant est défini par la classe `Dessin`. Une fois `F` affichée, grâce à la méthode `setVisible()`, la méthode `paint()` est exécutée automatiquement, et le triangle s'affiche.

La construction du sapin

Sachant maintenant afficher un simple triangle, nous pouvons construire le sapin par juxtaposition d'un ensemble de triangles. Pour réaliser le bon positionnement des triangles, utilisons la technique développée à la section « Les tableaux à deux dimensions » du chapitre 9, « Collectionner un nombre fixe d'objets ». Analysons la classe `Arbre`, qui reprend ce procédé :

```
import java.awt.*;
public class Arbre {
 private int [][] sapin ;
 private Color décoration;
 public Arbre(int nl, Color c) {
  int nc = 2*nl-1;
  décoration = c;
  sapin = new int[nl][nc];
  int milieu = sapin[0].length/2;
  for ( int j = 0 ; j < nl ; j++)
    for ( int i = -j; i <= j; i++)
      sapin[j][milieu+i] = (int ) (5*Math.random()+1);
 }
 public void dessine(Graphics g) {
  Color Vert = Color.green;
  for (int i = 0; i < sapin.length; i++) {
    for (int j = 0; j < sapin[0].length; j++) {
      switch(sapin[i][j]) {
        case 1 :  new Triangle(i, j, g, décoration);
                  break;
        case 2 :  Vert = Vert.brighter();
                  new Triangle(i, j, g, Vert);
                  break;
        case 3 :  Vert = Vert.darker();
                  new Triangle(i, j, g, Vert);
                  break;
```

Copyright © 2016 Eyrolles.

```
            case 4 :  Vert = Vert.brighter();
                      new Triangle(i, j, g, Vert);
                      break;
            case 5 :  Vert = Vert.darker();
                      new Triangle(i, j, g, Vert);
                      break;
          }
        }
      }
    }
  }
```

La classe `Arbre` est composée de deux données, le tableau d'entiers à deux dimensions `sapin` et la couleur `décoration`. Elle comporte en outre les deux méthodes suivantes :

- Le constructeur `Arbre()`, qui initialise la couleur de la `décoration`, et le tableau `sapin`, qui utilise la même technique que le sapin affiché en caractères graphiques. Les paramètres du constructeur rendent possible la création de sapins de taille et de couleur différentes.
- La méthode `dessine()` qui, en parcourant le tableau `sapin`, crée un triangle à l'aide du constructeur de la classe `Triangle` pour toute valeur `sapin[i][j]` différente de `0`. Grâce aux paramètres du nouveau constructeur `Triangle()`, le triangle est affiché à l'écran en fonction de sa position dans le tableau (indices `i` et `j`) et de la valeur du tableau (`sapin[i][j]`).

Remarque

Les méthodes `darker()` et `brighter()`, permettent de foncer ou d'éclaircir la couleur sur laquelle la méthode est appliquée. Grâce à ces méthodes, le sapin n'apparaît pas d'un vert uniforme.

Les paramètres du constructeur de la classe `Triangle` sont donc modifiés de façon à ne plus afficher un seul triangle vert au centre de la fenêtre mais un triangle d'une couleur et d'une position données. Pour réaliser cela, le constructeur est défini avec un ensemble de paramètres caractérisant la position en `x` et `y` à l'écran, ainsi que la couleur d'affichage du triangle. Examinons cette modification dans la classe `Triangle` ci-dessous :

```
import java.awt.*;
public class Triangle {
 private int pX = Fenetre.LG/2-50;
 private int pY = Fenetre.HT/2-50;
 private int [] xPoints = {0, 10, -10};
 private int [] yPoints = {-10, 10, 10};
 private int nPoints = 3;
 public Triangle(int lig, int col, Graphics g, Color c) {
```

Copyright © 2016 Eyrolles.

```
    for (int i = 0; i < nPoints; i++) {
      xPoints[i] = xPoints[i]+(5*col) + pX;
      yPoints[i] = yPoints[i]+(15*lig) + pY;
    }
    g.setColor(c);
    g.fillPolygon(xPoints, yPoints, nPoints);
  }
}
```

Chaque triangle affiché ne se trouve plus au centre de la fenêtre mais à une position spécifiée en paramètre. Cette position est déterminée par les éléments suivants :

- un point de référence (`pX, pY`) défini en fonction de la taille de la fenêtre ;
- la position (`i, j`) du triangle dans le tableau `sapin`.

Ces valeurs sont transmises au constructeur grâce aux paramètres `col` et `lig`. Ces valeurs étant connues, les sommets du polygone sont calculés de façon à afficher ce dernier au bon endroit à l'écran. Comme tous les triangles prennent un certain espace en hauteur et en largeur, il est nécessaire d'appliquer un coefficient (`5` et `10`) aux indices `lig` et `col` pour que chaque triangle ne se superpose pas trop à ses voisins.

Question Que se passe-t-il si l'on initialise les variables `px` et `py` à `0` ?

Réponse L'affichage d'un élément graphique est réalisé par rapport à un point d'origine placé conventionnellement en haut à gauche de la fenêtre. Le point de référence (`pX, pY`) permet de déplacer cette origine au centre de la fenêtre. En les initialisant à `0`, le sapin ne s'affiche plus au centre, mais dans le coin supérieur gauche de la fenêtre.

Le dessin du sapin

Pour que le sapin construit en mémoire s'affiche, ce dernier doit être placé dans la fenêtre de dessin. C'est ce que réalise la classe `Dessin` suivante :

```
import java.awt.*;
public class Dessin extends Canvas {
 private Color couleur = Color.green;
 public final static Color couleurFond = Color.white;
 private Arbre A;
 public Dessin()  {
  setBackground(couleurFond);
  setForeground(couleur);
  setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
  A = new Arbre(8, Color.yellow);
 }
```

Copyright © 2016 Eyrolles.

```
  public void paint (Graphics g)  {
   A.dessine(g);
  }
}
```

Cette classe reprend en grande partie l'architecture de la classe `Dessin`, décrite à la section « Dessiner un triangle », au début de ce chapitre. Pour remplacer le triangle par un sapin, le constructeur `Dessin()` crée en mémoire un objet `A` de type `Arbre`. La méthode `paint()` appelle ensuite la méthode `dessine()` par l'intermédiaire de l'objet `A` pour l'afficher à l'écran.

Remarque

Le fichier correspondant à l'application `Fenetre` n'a pas besoin d'être modifié. Lorsque l'application est exécutée, une fenêtre s'affiche avec son composant de type `Dessin`. Ce dernier crée en mémoire un objet `A` de type `Arbre`, puis la méthode `paint()` est automatiquement appelée par l'interpréteur. Le sapin est alors affiché.

Question

Pourquoi placer l'instruction `A = new Arbre(8, Color.yellow) ;` dans le constructeur `Dessin()` et non dans la méthode `paint()` ?

Réponse

L'instruction `A = new Arbre(8, Color.yellow) ;` créé en mémoire les données numériques qui vont être ensuite utilisées pour être affichées. Il est donc logique de réaliser la création en mémoire de valeurs numériques dans un constructeur.

En outre, si l'instruction `A = new Arbre(8, Color.yellow) ;` est placée dans la méthode `paint()`, de nouvelles données sont créées à chaque fois que la fenêtre doit être affichée. Ainsi, déplacer la fenêtre oblige l'interpréteur à afficher la fenêtre à l'endroit désigné. Un sapin différent du précédent s'affiche puisque de nouvelles valeurs ont été créées. Ce nouvel affichage est incorrect puisque le déplacement d'une fenêtre n'entraîne jamais la modification de son contenu.

Les éléments de communication graphique

Outre les composants d'affichage tels que les `Frame` et les `Canvas`, la bibliothèque AWT propose des outils de communication graphique, comme les boutons et les menus.

Ces outils offrent la possibilité d'écrire des applications munies d'une **interface graphique** réellement interactive. L'utilisateur manipule directement les objets proposés par l'interface, et cette dernière réagit en fonction des actions de l'utilisateur. Puisqu'il n'est pas possible de savoir à l'avance quel objet va être manipulé, chaque composant doit être programmé de façon à réagir directement aux manipulations de l'utilisateur. Chaque manipulation est considérée comme un **événement**, auquel est associé un traitement, c'est-à-dire une **action**.

Afin d'étudier ces différents concepts, nous allons améliorer l'application du sapin de Noël en y insérant deux boutons : un premier bouton pour afficher un sapin avec de nouvelles décorations et un second pour quitter l'application.

Copyright © 2016 Eyrolles.

Les boutons

Les boutons sont les composants de communication les plus utilisés pour créer des interfaces graphiques. Grâce à eux, par un simple clic, l'utilisateur valide son souhait de voir réaliser le traitement proposé par le bouton.

Les boutons sont définis dans la bibliothèque AWT par la classe `Button`. Pour afficher un bouton, il suffit de l'ajouter à une fenêtre, comme nous l'avons déjà fait pour dessiner un objet `Canvas`. Examinons la classe `Fenetre`, dans laquelle nous allons insérer deux boutons :

```
import java.awt.*;
public class Fenetre {
public  final static int HT = 300;
public   final static int LG = 300;
 public static void main(String [] arg) {
  Frame F = new Frame();
  // …
  F.add(new Dessin());
  F.add(new Button("Nouveau"));
  F.add(new Button("Quitter"));
  F.setVisible(true);
 }
}
```

Dans cet exemple, deux boutons, portant les noms `"Quitter"` et `"Nouveau"`, sont ajoutés à la fenêtre `F` grâce à la méthode `add()`. Lorsque le programme est exécuté, l'affichage résultant est celui illustré à la figure 11-3.

Chaque composant (`Canvas` et `Button`) est ajouté à la fenêtre, sans que soient spécifiés ni sa position, ni sa taille. Dans cette situation, l'interpréteur affiche les composants en les superposant dans leur ordre d'arrivée. Le dernier bouton, `"Quitter"`, cache par conséquent le composant `Dessin` ainsi que le bouton `Nouveau`.

Figure 11-3 *Les composants graphiques s'affichent en se superposant les uns aux autres. C'est pourquoi le dernier bouton cache les autres composants.*

Copyright © 2016 Eyrolles.

Les conteneurs

Pour corriger cette erreur, il convient, lorsque vous souhaitez afficher plusieurs composants graphiques, de placer ces derniers à l'intérieur d'un conteneur (en anglais *container*).

Remarque

Un conteneur est une sorte de boîte qui contient tous les éléments de communication utilisés dans l'application. La plupart des boîtes à outils proposées dans les logiciels récents sont des conteneurs. Notez que seules les `Frame` ne peuvent être placées dans un conteneur.

Un conteneur est défini par la classe `Panel` du package `java.awt`. Examinons comment l'utiliser dans le programme suivant :

```
import java.awt.*;
public class DesBoutons extends Panel {
 public DesBoutons() {
  // initialisation
  setBackground(Color.lightGray);
  // Les boutons
  Button bNouveau = new Button ("Nouveau");
  this.add(bNouveau);
  Button bQuitter = new Button ("Quitter");
  this.add(bQuitter);
 }
}
```

La classe `DesBoutons` est définie comme classe héritant de la classe `Panel` (`DesBoutons extends Panel`). Elle est composée d'un constructeur qui crée en mémoire deux boutons `bNouveau` et `bQuitter`, et les ajoute ensuite au conteneur grâce à la méthode `add()`. Par défaut, les boutons sont affichés au centre du `Panel` par ordre d'arrivée.

Remarque

L'application du terme `this` aux méthodes `add()` est facultatif. Ce terme indique à l'interpréteur qu'il doit ajouter ces objets (les boutons) à l'objet qu'il est en train de construire, c'est-à-dire au `Panel` nommé `DesBoutons`. L'expression `this` représente l'objet qui se construit en mémoire.

Une fois défini, le conteneur doit être ajouté à la fenêtre. Pour éviter toute superposition du conteneur à l'objet `Canvas`, il est possible d'indiquer à l'interpréteur comment afficher les éléments les uns par rapport aux autres. Utilisons à cette fin les termes `"South"`, `"North"`, `"Center"`, `"East"` et `"West"` en paramètre de la méthode `add()`.

Copyright © 2016 Eyrolles.

Le programme `Fenetre` ci-dessous utilise cette technique pour afficher correctement les deux boutons :

```
import java.awt.*;
public class Fenetre{
 //..
 public static void main(String [] arg) {
  Frame F = new Frame();
  //..
  F.add(new Dessin(), "Center");
  F.add(new DesBoutons(), "South");
  F.setVisible(true) ;
 }
}
```

Grâce aux paramètres `"Center"` et `"South"`, les composants s'affichent correctement dans la fenêtre de dessin, au-dessus de la boite à boutons, comme illustré à la figure 11-4.

Notre application possède maintenant deux boutons. Pourtant, lorsque l'utilisateur clique sur l'un ou l'autre de ces boutons, rien ne se passe : l'affichage de nouveaux sapins n'est pas effectué, et il n'est pas non plus possible de quitter l'application en cliquant sur le bouton `"Quitter"`.

C'est qu'il ne suffit pas d'afficher un bouton avec un texte correspondant à l'action souhaitée pour voir cette action se réaliser. La classe `Button` ne fait que définir les attributs graphiques des boutons. Pour associer un bouton à une action, il faut encore gérer les événements.

Figure 11-4 *Une fois la position des composants définie par rapport aux bords de la fenêtre principale, chaque composant s'affiche correctement.*

Copyright © 2016 Eyrolles.

Les événements

En langage Java, la gestion des événements est réalisée par l'intermédiaire d'objets spécifiques, appelés écouteurs (en anglais *listener*). De façon simplifiée, on peut dire que, lorsque l'utilisateur clique sur un bouton ou sur une commande de menu, le composant concerné émet un événement à l'attention de l'écouteur.

Le traitement de cet événement est réalisé par l'écouteur d'événement (`Eventlistener`) et non pas par le composant lui-même. Le langage Java gère les événements en suivant un modèle dit « par délégation » (en anglais *delegation model*), le traitement de l'événement étant délégué à un autre composant que celui qui l'a perçu.

Les types d'événements

Chaque composant graphique émet un événement propre à sa classe, et il existe donc plusieurs types d'événements. On distingue les « événements de bas niveau » et les « événements de haut niveau ».

Événements de bas niveau

Les événements de bas niveau sont les événements créés à partir de la souris, du clavier ou d'une fenêtre. Le tableau suivant résume les types d'événements de bas niveau les plus utilisés.

Écouteur	Comportement à programmer	
`MouseListener` écoute les événements liés à la souris.	`mousePressed(MouseEvent)`	Appelé lors d'une pression sur un bouton de la souris.
	`mouseReleased(MouseEvent)`	Appelé lorsqu'un bouton de la souris est relâché.
	`mouseExited(MouseEvent)`	Appelé lorsque la souris sort de la fenêtre.
	`mouseEntered(MouseEvent)`	Appelé lorsque la souris entre dans la fenêtre.
	`mouseClicked(MouseEvent)`	Appelé lors d'un simple clic de souris.
`MouseMotionListener` écoute les événements liés à la souris lorsqu'elle se déplace.	`mouseDragged(MouseEvent)`	Appelé lorsque la souris est déplacée, bouton enfoncé.
	`mouseMoved(MouseEvent)`	Appelé lorsque la souris est déplacée, bouton relâché.

Copyright © 2016 Eyrolles.

`WindowListener` écoute les événements liés à la fenêtre.	`windowClosing(WindowEvent)`	Appelé lorsque la fenêtre est en train de se fermer.
	`windowClosed(WindowEvent)`	Appelé lorsque la fenêtre est fermée.
	`windowOpened(WindowEvent)`	Appelé lors de l'ouverture de la fenêtre.
	`windowIconified(WindowEvent)`	Appelé lorsque la fenêtre est mise sous forme d'icône.
	`windowDeiconified(WindowEvent)`	Appelé lorsque l'icône est agrandie à la taille de la fenêtre.
	`windowActived(WindowEvent)`	Appelé lorsque la fenêtre est activée et reçoit les événements du clavier.
	`windowDeactived(WindowEvent)`	Appelé lorsque la fenêtre est désactivée et perd les événements du clavier.

Pour gérer un événement lié à la souris, par exemple, il convient de définir un écouteur de type `MouseListener` et de décrire le comportement de l'application pour chaque méthode associée à cet écouteur.

Événements de haut niveau

Les événements de haut niveau sont liés, non plus aux comportements du composant graphique, mais à ses actions possibles. Ainsi, un clic de souris sur un bouton ne génère plus un événement spécifique du composant mais un comportement défini par le programmeur.

Écouteur	Comportement à programmer	
`ActionListener` écoute les événements d'action.	`actionPerformed(ActionEvent)`	Appelé lorsqu'une action est émise.

Ainsi, une action est associée à un bouton en définissant un écouteur d'action qui, par l'intermédiaire de la méthode `actionPerformed()`, réalise l'action souhaitée.

Exemple : associer un bouton à une action

Lorsque l'utilisateur clique sur les boutons `"Nouveau"` ou `"Quitter"` de l'application développée dans ce chapitre, il souhaite voir se réaliser deux actions distinctes : soit l'affichage d'un nouveau sapin, soit la fermeture de la fenêtre.

Copyright © 2016 Eyrolles.

Chaque clic de souris sur un bouton est associé à une action spécifique, qui utilise un événement de haut niveau. Par conséquent, chaque bouton doit être muni d'un écouteur d'action. Cela est réalisé dans la classe `DesBoutons`, comme ci-dessous :

```
import java.awt.*;
import java.awt.event.*;
public class DesBoutons extends Panel {
 public DesBoutons(Dessin d)  {
  //…
  Button bNouveau = new Button ("Nouveau");
  Button bQuitter = new Button ("Quitter");
  bNouveau.addActionListener(new GestionAction(1, d));
  this.add(bNouveau);
  bQuitter.addActionListener(new GestionAction(2, d));
  this.add(bQuitter);
 }
}
```

Remarque L'instruction d'import (`import java.awt.event.*;`) indique au compilateur le package où sont définies les méthodes de gestion des événements utilisées dans la classe.

La mise en place des écouteurs d'actions est réalisée grâce à la méthode `addAction Listener()`.

Le constructeur `GestionAction()`, placé en paramètre de la méthode `addAction Listener()`, permet de préciser quel comportement doit adopter l'application en fonction du bouton qui émet l'événement. En effet, les paramètres de ce constructeur transmettent à la fois une valeur entière différente (`1` ou `2`) suivant le bouton émetteur (`bNouveau` ou `bQuitter`) et l'adresse de l'objet (`d`) sur lequel est dessiné le sapin. Examinons comment sont gérés ces paramètres dans la classe `GestionAction` :

```
import java.awt.*;
import java.awt.event.*;
public class GestionAction implements ActionListener {
 private int n;
 private Dessin d;
 public GestionAction( int n, Dessin d) {
  this.n = n;
  this.d = d;
 }
 public void actionPerformed(ActionEvent e){
  switch (n) {
    case 1 :  d.nouveau();
```

Copyright © 2016 Eyrolles.

```
            break;
      case 2 :  System.exit(0);
            break;
    }
  }
}
```

La classe `GestionAction` fait appel à plusieurs notions importantes, développées dans les sections qui suivent.

Le terme implements

La classe `GestionAction` implémente la classe `ActionListener` (`GestionAction implements ActionListener`) ; elle n'en hérite pas.

Comme nous avons pu l'observer à la section « Les interfaces » du chapitre 8 « Les principes du concept objet », les méthodes définies par un `listener` ne peuvent pas être prédéfinies par le langage Java. Une action, c'est-à-dire un comportement associé à un bouton, ne peut être décrite que par le programmeur, selon la façon dont il conçoit son application. On dit alors que la classe `ActionListener`, ainsi que tous les autres `listener`, est une **interface** qui définit simplement les différents modes de comportement.

Lorsqu'une classe dérive d'une interface, le terme `implements` doit être utilisé au lieu du terme `extends`.

De plus, lorsqu'une classe implémente une interface, le compilateur Java exige de décrire l'intégralité des méthodes définies par l'interface. En effet, dans le cas où l'une des méthodes n'est pas définie, le compilateur indique une erreur en précisant le nom de la méthode manquante.

Dans notre exemple, l'interface `ActionListener` ne définit qu'une seule méthode (`actionPerformed()`). Nous n'avons donc aucune difficulté à décrire l'intégralité des méthodes proposées par cette interface.

Le terme this

La classe `GestionAction` possède deux variables d'instance `n` et `d`, de façon à mémoriser la valeur respective transmise par les boutons, ainsi que l'adresse de la zone graphique où le sapin est dessiné. Ces valeurs sont initialisées par l'intermédiaire des paramètres du constructeur `GestionAction()` qui sont également nommés `n` et `d`.

Pour éviter toute confusion entre les données de la classe et les paramètres du constructeur, il est nécessaire d'employer le terme `this`. Ce terme appliqué à `n` et `d`, précise au compilateur qu'il s'agit des variables de l'instance qui se construit. Sans `this`, les mêmes noms de variables correspondraient aux paramètres du constructeur.

Copyright © 2016 Eyrolles.

La méthode actionPerformed()

Une fois les données initialisées, la méthode `actionPerformed()` est automatiquement exécutée par l'interpréteur, lorsqu'une action est émise par l'un des boutons suite à un clic de l'utilisateur. Cette dernière réalise alors, suivant la valeur transmise au constructeur (`1` ou `2`), soit la sortie du programme, soit l'affichage d'un nouveau sapin, grâce à la méthode `nouveau()` (à insérer dans la classe `Dessin`) décrite ci-dessous :

```
public void nouveau() {
  A = new Arbre(6, Color.red);
  repaint();
}
```

À l'exécution de cette méthode, l'objet `A` est recalculé à l'aide du constructeur `Arbre()`. Ensuite, la méthode `repaint()` efface automatiquement la zone d'affichage `d` sur laquelle la méthode est appliquée et appelle la méthode `paint()` définie dans la classe `Dessin`.

La donnée Dessin d

Le bouton `bNouveau` a une incidence sur la zone de dessin puisqu'un nouveau sapin doit être affiché dans cette zone suite à un clic sur le bouton. Cet effet est réalisé par le biais de la méthode `nouveau()`, appliquée à l'objet `d` de type `Dessin` dans la méthode `actionPerformed()`. L'objet `d` est déclaré comme variable d'instance de la classe `GestionAction`. Il contient l'adresse de la zone d'affichage créée dans l'application `Fenetre`, comme l'illustre l'extrait de programme suivant :

```
public class Fenetre {
//…
 public static void main(String [] arg) {
  Frame F = new Frame();
  //…
  Dessin page = new Dessin();
  F.add(page, "Center");
  F.add(new DesBoutons(page), "South");
}
}
```

La construction de l'objet `page` a pour résultat de stocker en mémoire l'adresse du composant graphique de type `Canvas`. Une fois cette adresse transmise au constructeur de la classe `DesBoutons`, ce dernier peut transmettre à son tour l'adresse de l'objet `page` au constructeur de la classe `GestionAction` par l'intermédiaire du paramètre formel `d` de type `Dessin`. Grâce à la transmission de l'adresse de la zone graphique en paramètre des constructeurs, les nouveaux sapins s'affichent dans le composant graphique approprié.

Copyright © 2016 Eyrolles.

Exemple : fermer une fenêtre

Pour fermer une fenêtre en cliquant directement sur l'icône de fermeture de la fenêtre située dans la barre de titre, l'événement « clic sur le bouton de fermeture de la fenêtre » doit être associé à l'instruction qui permet de stopper l'exécution du programme. Comme le précise le tableau de description des écouteurs, l'événement « clic sur le bouton de fermeture de la fenêtre » est un événement de bas niveau, géré par l'écouteur `WindowListener`.

En effet, lorsque l'utilisateur ferme la fenêtre par l'intermédiaire de l'icône appropriée, ce dernier émet un événement de fermeture de fenêtre. En recevant cet événement, l'écouteur des événements de fenêtre (`WindowListener`) exécute automatiquement la méthode `windowClosing()`.

Par conséquent, le programme qui réalise la fermeture d'une fenêtre doit effectuer les deux opérations suivantes :

- placer un écouteur d'événement de fenêtre dans le composant graphique autorisé à être fermé de la sorte ;
- programmer la méthode `windowClosing()` en y insérant l'instruction `System.exit(0)` de façon à sortir de l'application.

Placer un écouteur d'événement de fenêtre

Le premier point est réalisé dans la classe `Fenetre` de la façon suivante :

```
import java.awt.*;
public class Fenetre {
 //..
 public static void main(String [] arg) {
  Frame F = new Frame();
  //…
  F.addWindowListener(new GestionFenetre());
  F.setVisible(true);
 }
}
```

Une fois la méthode `addWindowListener()` appliquée à la fenêtre `F`, cette dernière est à même d'écouter les événements émis par ses propres composants.

Programmer la méthode windowClosing()

Le second point est résolu grâce à l'appel du constructeur `GestionFenetre()` en paramètre de la méthode `addWindowListener()`, ce dernier définissant le comportement adopté en face de l'événement entendu. Examinons plus attentivement la classe `GestionFenetre` :

```
import java.awt.event.*;
public class GestionFenetre extends WindowAdapter{
 public void windowClosing(WindowEvent e){
```

Copyright © 2016 Eyrolles.

```
        System.exit(0);
    }
}
```

Remarque La classe `GestionFenetre` hérite de la classe `WindowAdapter` au lieu d'implémenter l'interface `WindowListener`.

En effet, l'écouteur `WindowListener` possède sept comportements spécifiques (voir précédemment le tableau des événements de bas niveau). Si nous implémentons directement cette interface, comme nous l'avons fait pour `ActionListener`, nous sommes contraints par le compilateur Java à définir l'ensemble des sept comportements.

Or, pour fermer la fenêtre, un seul comportement est à retenir : celui défini par la méthode `windowClosing()`. En utilisant un `Adapter`, plutôt qu'un `Listener`, nous n'avons plus l'obligation de définir l'intégralité des sept comportements mais uniquement la ou les méthodes de notre choix. Pour notre exemple, seule la méthode `windowClosing()` nous intéresse. C'est pourquoi elle seule est décrite dans la classe `GestionFenetre`.

Ainsi, lorsque l'utilisateur clique sur l'icône de fermeture de la fenêtre, un événement de fermeture de fenêtre est émis. L'événement est capté et traité par l'écouteur `WindowListener`, qui exécute automatiquement la méthode `windowClosing()`. Celle-ci termine l'exécution de l'application grâce à l'instruction `System.exit(0)`.

Quelques principes

L'analyse des exemples précédents montre que la gestion d'un événement quel qu'il soit passe par les trois étapes suivantes :

- Déterminer le composant qui émet l'événement et lui associer un écouteur. Cette association est réalisée par une méthode ayant pour nom `addxxxListener()`, où `xxx` représente le composant émetteur (`Window`, `Mouse`, etc.).
- Créer une classe `gestionDelEvenement` qui implémente l'interface `xxxListener` (`implements`) ou dérive de la classe `xxxAdapter` (`extends`), selon que vous souhaitiez traiter tout ou partie des méthodes proposées par l'écouteur.
- Développer les méthodes souhaitées, c'est-à-dire décrire les instructions composant les méthodes définies par l'interface utilisée.

De l'AWT à Swing

La bibliothèque graphique Swing a été introduite à partir de la version 1.2 du JDK. Les composants de cette bibliothèque diffèrent légèrement des composants de la bibliothèque

Copyright © 2016 Eyrolles.

AWT. Ils ont l'avantage d'être écrits, pour la plus grande part, en langage Java et non à l'aide des fonctions internes de l'ordinateur. Cette particularité fait que l'on peut, par exemple, créer ses propres boutons ou modifier le style de l'interface, sans avoir à redémarrer le programme.

En effet, lorsqu'un bouton de la bibliothèque Swing est affiché à l'écran, il est entièrement dessiné par l'interpréteur Java et non par le système de l'ordinateur. Par conséquent, un programme fonctionnant sur une station Unix peut dessiner un bouton à la façon Macintosh en utilisant la méthode Java appropriée.

Par ailleurs, les composants graphiques de la bibliothèque Swing utilisent très peu de place mémoire. Ils sont appelés composants légers. À l'opposé, il existe des composants lourds développés à l'aide des fonctions du système dont ils dépendent. Ces composants sont incontournables puisque *in fine*, un programme doit bien communiquer avec le système de l'ordinateur. Avec la bibliothèque Swing, le nombre de ces composants lourds est très réduit. Ce sont les fenêtres du type `JFrame`, `JApplet`, `JDialog` et `JWindow`.

Par convention, les objets de la bibliothèque Swing portent le même nom que leurs homologues de la bibliothèque AWT, précédés de la lettre `J`. Ainsi l'objet `Frame` du package AWT devient `JFrame` dans le package Swing.

La première fenêtre affichée est un composant qui dépend du système. Elle correspond toujours à un composant lourd. En revanche, les composants graphiques tels que les boutons, les menus ou les sous-fenêtres contenus dans cette fenêtre n'ont pas besoin de communiquer avec le système, il leur suffit de communiquer avec la fenêtre initiale. C'est pourquoi la plus grande partie des composants utilisés pour construire une interface graphique sont des composants légers.

Un sapin en Swing

Pour comprendre comment utiliser les composants de la bibliothèque Swing, transformons tous les objets graphiques de la classe `Fenetre` décrite à la section « Exemple : Dessiner un sapin de Noël » en leur équivalent, dans la bibliothèque Swing.

La classe FenetreSwing

Examinons la nouvelle classe, `FenetreSwing` :

```
import javax.swing.* ;
public class FenetreSwing {
 public  final static int HT = 300 ;
 public   final static int LG = 300 ;
 public static void main(String [] arg) {
```

Copyright © 2016 Eyrolles.

```
    JFrame F = new JFrame("Une fenetre en Swing") ;
    F.setSize(HT, LG) ;            // taille de la fenêtre
    F.setVisible(true);
  }
}
```

En premier lieu, nous notons la nouvelle instruction d'import du package Swing (`import javax.swing.*;`). Sans cette instruction, le compilateur Java ne peut pas comprendre la déclaration des objets définis par cette bibliothèque.

Remarque

Le package Swing a porté plusieurs noms en fonction des différentes versions du JDK. Certaines versions bêta de Java ont porté le nom de `java.awt.swing`. Aujourd'hui, la convention adoptée par Sun fait que la bibliothèque Swing est considérée comme une extension au langage Java, d'où le `'x'` (comme extension) de `javax.swing`.

Ensuite, la fenêtre principale est construite en mémoire grâce à la déclaration de l'objet `F` de type `JFrame` (`JFrame F = new JFrame("Une fenetre en Swing") ;`). Le paramètre du constructeur `JFrame()` donne un titre à la fenêtre qui s'affiche à l'appel de la méthode `setVisible`, en dernière ligne du programme (`F.setVisible(true) ;`).

La fenêtre ainsi affichée se présente sous la forme d'une fenêtre classique telle que nous l'avons déjà vu en section « Les fenêtres » de ce chapitre.

Cependant, il s'agit d'une `JFrame` considérée en Swing comme un conteneur de composants graphiques, c'est-à-dire un objet qui peut contenir d'autres objets graphiques.

Afin d'organiser au mieux les composants qu'elle contient, une `JFrame` est décomposée en deux entités : la barre de menus (`Menu Bar`) et le panneau de contenus (`Content Pane`), ainsi que le montre la figure 11.5.

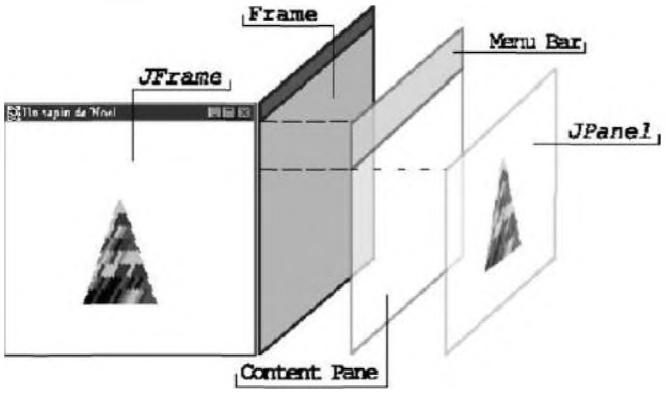


Figure 11-5 *Une JFrame est composée d'une Frame qui se décompose à son tour en une barre de menus (Menu Bar) et panneau de contenu (Content Pane).*

Copyright © 2016 Eyrolles.

Une `JFrame` est donc considérée comme un conteneur de haut niveau se situant à la racine du panneau de contenus et de la barre de menu, comme le montre la figure 11-6.

La classe SapinSwing

L'affichage du sapin s'effectue à l'intérieur du panneau de contenus, comme le montre le code source suivant :

```
import javax.swing.* ;

public class SapinSwing {
 public  final static int HT = 300 ;
 public   final static int LG = 300 ;
 public static void main(String [] arg) {
  JFrame F = new JFrame("Un sapin Swing") ;
  F.setSize(HT, LG) ;          // taille de la fenêtre
  F.getContentPane().add(new DessinSwing()) ;
  F.setVisible(true) ;         // affiche la fenêtre
}
```

Pour afficher le sapin, nous devons appeler le constructeur de la classe `DessinSwing` afin de le créer dans le panneau de contenu de la `JFrame`. Pour ce faire, nous faisons appel à la méthode `getContentPane()`, qui a pour rôle :

- de déterminer les éléments déjà définis dans le panneau de contenus ;
- puis d'ajouter l'élément placé en paramètre `(new DessinSwing())` dans le panneau de contenus.

De cette façon, aucun composant ne peut être omis, ni placé de façon incorrecte.

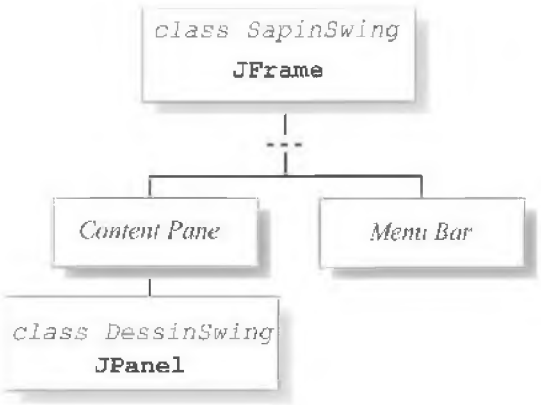


Figure 11-6 *Dans la hiérarchie des composants graphiques, le composant Jpanel est placé dans le panneau de contenu (Content Pane) grâce à la méthode getContentPane().add(new DessinSwing()).*

Copyright © 2016 Eyrolles.

La classe DessinSwing

Examinons à présent la classe `DessinSwing` qui permet de dessiner le sapin dans le panneau de contenu.

```
import java.awt.* ;
import javax.swing.* ;
public class DessinSwing extends JPanel {
 private Color couleur = Color.green ;
 public final static Color couleurFond = Color.white ;
 private Arbre A ;
 public Dessin()  {
  setBackground(couleurFond) ;
  setForeground(couleur) ;
  setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR)) ;
  A = new Arbre(8, Color.yellow) ;
 }
 public void paintComponent (Graphics g)  {
  super.paintComponent(g) ;
  A.dessine(g) ;
 }
}
```

La classe `DessinSwing` hérite de la classe `JPanel` qui remplace entre autres, la structure `Canvas` de la bibliothèque AWT. `JPanel` est un composant graphique qui a pour avantage de correspondre à un composant léger. Le `JPanel` est utilisé pour contenir la plus grande partie des composants graphiques tels que les boutons, les cases à cocher, etc. Il a été conçu pour faciliter le positionnement des composants graphiques qu'il contient.

Remarque La méthode d'affichage du composant n'est plus la méthode `paint()` utilisée dans la classe `Dessin` décrite plus haut au paragraphe « Le dessin du sapin ». En effet, les objets de type `JPanel` sont dessinés par la méthode `paintComponent()`.

Cette méthode fait obligatoirement appel au constructeur de la classe supérieure par l'intermédiaire de l'instruction `super.paintComponent()`, afin d'afficher les autres composants éventuellement placés dans le panneau de contenu.

Modifier le modèle de présentation de l'interface

Nous l'avons écrit au tout début de cette section, l'intérêt de la bibliothèque Swing est qu'elle est écrite en Java. La plus grande partie des composants graphiques sont donc affichés par l'interpréteur Java. Il devient ainsi facile de modifier l'apparence (en anglais *Look and Feel*) d'un bouton ou d'un menu, en cours d'exécution du programme.

Copyright © 2016 Eyrolles.

Question Que se passe-t-il si le constructeur de la classe supérieure `super.paintComponent()` n'est pas appelé lors de l'affichage du composant `Jpanel` ?

Réponse En omettant l'appel du constructeur de la classe supérieure, l'interpréteur n'est pas à même de gérer correctement l'affichage du panneau de contenu. En effet, il ne peut savoir quels composants graphiques ont pu être placés dans la structure `ContentPane`, en amont ou en aval de l'appel de la méthode `paintComponent()`.

Examinons comment réaliser cela, à travers l'exemple suivant :

Il s'agit d'ajouter un bouton à l'interface créée au paragraphe « Exemple : Associer un bouton à une action ». Ce nouveau bouton a pour rôle de modifier l'apparence des boutons de l'application lorsque l'on clique dessus.

Pour réaliser cela, nous devons :

- Ajouter un nouveau bouton (voir section « La classe DesBoutonsSwing »).
- Associer une action au bouton (voir section « La classe GestionAction »).
- Afficher les boutons dans la fenêtre principale (voir section « La classe SapinSwing »).

La classe DesBoutonsSwing

```
import java.awt.* ;
import javax.swing.* ;
import java.awt.event.* ;
public class DesBoutonsSwing extends JPanel {
  public DesBoutons(Dessin d, JFrame j)  {
    setBackground(Color.lightGray) ;
    // Les boutons
    JButton bPeindre = new JButton ("Nouveau") ;
    bPeindre.addActionListener(new GestionAction(1, d, j)) ;
    this.add(bPeindre) ;
    JButton bModèle = new JButton ("Modele") ;
    bModèle.addActionListener(new GestionAction(3, d, j)) ;
    this.add(bModèle) ;
    JButton bQuitter = new JButton ("Quitter") ;
    bQuitter.addActionListener(new GestionAction(2, d, j)) ;
    this.add(bQuitter) ;
  }
}
```

Afin d'afficher les trois boutons en bas de la fenêtre principale, nous allons les placer dans un conteneur de type `JPanel` (`public class DesBoutonsSwing extends JPanel`). De cette façon, nous n'aurons qu'à placer ensuite ce conteneur dans le panneau de contenu en

Copyright © 2016 Eyrolles.

dessous du conteneur qui affiche le sapin (voir la classe `SapinSwing` ci-dessous et figure 11.7).

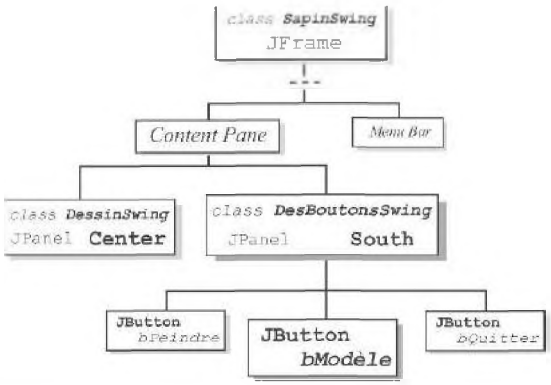


Figure 11-7 *Les trois boutons Nouveau (bPeindre), Modèle (bModèle) et Quitter (bQuitter) sont placés dans un conteneur de type JPanel situé au même niveau que le conteneur qui affiche la sapin. Ces deux JPanel sont placés respectivement au sud et au centre, dans le panneau de contenu.*

En Swing, les boutons ne s'appellent plus `Button` mais `JButton`,. Ils fonctionnent cependant de la même façon. L'instruction `JButton bModèle = new JButton ("Modele") ;` crée en mémoire un bouton portant le nom `bModèle`.

Une fois créé, il est nécessaire d'associer au bouton un événement grâce à l'instruction `bModèle.addActionListener(new GestionAction(3, d, j)) ;` qui fait appel au constructeur de la classe `GestionAction`.

La classe GestionAction

La classe définissant toutes les actions associées aux différents boutons doit décrire de quelle façon l'apparence des boutons doit être modifiée. Examinons la méthode `gestionModèle()` qui est appelée lorsqu'on clique sur le bouton `Modèle` :

```
import java.awt.* ;
import java.awt.event.* ;
import javax.swing.* ;
import javax.swing.event.* ;
public class GestionAction implements ActionListener {
 private int n ;
 private Dessin d ;
 private static int modèle = 0 ;
 private JFrame j ;
 public GestionAction( int n, Dessin d, JFrame j) {
  this.n = n ;
```

Copyright © 2016 Eyrolles.

```
 this.d = d ;
 this.j = j ;
}

public void actionPerformed(ActionEvent e) {
 switch (n)  {
   case 1 : d.nouveau() ;
   break ;
   case 2 : System.exit(0) ;
   break ;
   case 3 : gestionModèle() ;
   break ;
  }
}
private void gestionModèle() {
 String LAndF = "com.sun.java.swing.plaf.motif.MotifLookAndFeel" ;
 try {
  switch(modèle) {
   case 0 :
    LAndF = "com.sun.java.swing.plaf.motif.MotifLookAndFeel" ;
    System.out.println(" Modele Motif ") ;
   break ;
   case 1 :
    LAndF = "javax.swing.plaf.metal.MetalLookAndFeel" ;
    System.out.println(" Modele Metal ") ;
   break ;
   case 2 :
    LAndF = "javax.swing.plaf.mac.MacLookAndFeel" ;
    System.out.println(" Modele Mac ") ;
   break ;
   case 3 :
    LAndF = "com.sun.java.swing.plaf.windows.WindowsLookAndFeel" ;
    System.out.println(" Modele Windows ") ;
   break ;
  }
  UIManager.setLookAndFeel(LAndF) ;
  SwingUtilities.updateComponentTreeUI(j) ;
 }
 catch(UnsupportedLookAndFeelException ex) {
  System.out.println("Exception : Modele non disponible") ;
 }
 catch(IllegalAccessException ex) {
  System.out.println("Exception : Modele non accessible") ;
 }
```

Copyright © 2016 Eyrolles.

```
        catch(ClassNotFoundException ex) {
         System.out.println("Exception : Modele non trouve") ;
        }
        catch(InstantiationException ex) {
         System.out.println("Exception : Modele non instanciable") ;
        }
        catch(Exception ex) {
         System.out.println("Exception : Erreur d'execution ") ;
        }
        modèle++ ;
        modèle = modèle % 4 ;
      }
    }
```

Lorsque l'utilisateur clique sur le bouton `Modele`, l'application affiche l'ensemble de ses boutons avec une nouvelle apparence.

Remarque

Il existe différentes apparences prédéfinies dans le package Swing que l'on trouve dans le répertoire `com.sun.swing.plaf`. Le terme `plaf` correspond aux initiales de l'expression *pluggable look and feel* qui peut se traduire littéralement par « apparence et sensation en prise directe » . Ce qui veut dire en français que l'apparence d'un bouton et la sensation réalisée par la modification de son apparence lors d'un clic sont modifiables « en direct », c'est-à-dire sans avoir à arrêter l'exécution du programme.

Ainsi, la méthode `gestionModèle()` mémorise selon la valeur de la variable `modèle` l'emplacement du modèle que l'on souhaite afficher.

Ensuite, l'instruction `UIManager.setLookAndFeel(LAndF) ;` fait appel à la méthode `setLookAndFeel()` de la classe `UIManager` afin de modifier le modèle de représentation des trois boutons. Cette modification est enfin prise en compte et affichée grâce à l'instruction `SwingUtilities.updateComponentTreeUI(j) ;`.

Remarque

La classe `UIManager` correspond, comme son nom l'indique, à la classe du gestionnaire des interfaces utilisateur. Elle regroupe l'ensemble des méthodes de gestion de la présentation (Look and Feel) des éléments d'une interface graphique.

Pour en savoir plus

La modification de l'apparence des boutons peut générer différentes erreurs qui sont détectées et gérées grâce aux mécanismes des instructions `try... catch` (voir au chapitre 10, « Collectionner un nombre indéterminé d'objets, la section « Les fichiers d'objets, gérer les exceptions »).

Copyright © 2016 Eyrolles.

La classe SapinSwing

Pour finir, les boutons sont ajoutés à la fenêtre principale comme suit :

```
import java.awt.* ;
import javax.swing.* ;
import java.awt.event.* ;
public class SapinSwing {
public  final static int HT = 300 ;
public   final static int LG = 300 ;
public  final static int X = 150 ;
public   final static int Y = 200 ;
  public static void main(String [] arg) {
  JFrame F = new JFrame("Un sapin de Noel en Swing") ;
    DessinSwing page = new DessinSwing() ;
    F.setBounds(X, Y, HT, LG) ;
    F.setBackground(Color.darkGray) ;
    F.addWindowListener(new GestionFenetre()) ;
    F.getContentPane().add(page, "Center") ;
    F.getContentPane().add(new DesBoutonsSwing(page, F), "South") ;
    F.setVisible(true) ;
  }
}
```

Comme pour la bibliothèque AWT, la méthode `add()` peut posséder deux arguments. Le premier correspond au composant à ajouter, le second à la place que ce dernier doit prendre dans le panneau de contenus. Ici, le panneau d'affichage du dessin est placé au centre (`Center`) alors que les boutons sont affichés en bas (`South`).

Résultat de l'exécution

L'exécution du programme `SapinSwing` a pour résultat l'affichage de la fenêtre représentée à la figure 11-8a. L'apparence correspond au `plaf` par défaut c'est-à-dire le Look and Feel `Metal`.

En cliquant sur le bouton `Modele`, la variable `modèle` est initialisée à `0` et le `plaf` est mis à jour et prend l'apparence `Motif` (voir figure 11-8c).

Le `plaf` représenté à la figure 11-8b correspond au Look and Feel `Windows`.

Observez que seule l'apparence des boutons est modifiée, le sapin n'est pas reconstruit. Il garde le même aspect.

Copyright © 2016 Eyrolles.

Figure 11-8a *Look and Feel Metal*

Figure 11-8b *Look and Feel Windows*

Figure 11-8c *Look and Feel Motif*

Copyright © 2016 Eyrolles.

Résumé

L'essentiel des composants graphiques développés par le langage Java est défini dans la bibliothèque AWT *(Abstract Windowing Toolkit)*.

Le support principal d'affichage d'une application graphique est la `Frame`. Cette dernière est composée des éléments suivants :

- une barre de titre possédant des boutons pour la fermeture, l'agrandissement et la mise en icône de la fenêtre ;
- des bords délimitant la zone d'exécution de l'application.

Pour dessiner ou afficher du texte dans une `Frame`, il convient d'utiliser des objets de type `Canvas` ou `TextArea`. Le contexte graphique définissant les attributs d'affichage, tels que la couleur, le type de fonte, etc., est géré par la classe `Graphics`.

Les interfaces graphiques sont construites à l'aide des éléments de communication graphique suivants :

- composants graphiques tels que boutons (`Button`) et menus ;
- événements associant par exemple, un clic de souris sur un bouton à une action.

On distingue les événements de **haut niveau** (un clic est associé à une action) et les événements de **bas niveau** (un clic sur un composant émet un événement propre à ce composant).

Depuis la version 1.2 du JDK, la bibliothèque Swing est proposée comme une extension graphique du langage Java. Les composants Swing ont l'avantage d'être écrits en Java, ce qui les rend plus légers en terme de mémoire. De plus, cette particularité fait que l'on peut modifier le style de l'interface sans avoir à redémarrer le programme. Ce mécanisme est réalisé à travers le concept du `plaf` (*pluggable look and feel*).

Par convention, les objets de la bibliothèque Swing portent le même nom que leurs homologues de la bibliothèque AWT, précédés de la lettre `J`. Ainsi l'objet `Frame` du package AWT devient `JFrame` dans le package Swing.

Copyright © 2016 Eyrolles.

Exercices

L'objectif des exercices 11.1 à 11.4 est d'améliorer le programme réalisé tout au long de ce chapitre. L'interface graphique à construire propose une case à cocher (en anglais *check box*) permettant de répondre aux conditions suivantes :

- Si la case `Taille fixe` est cochée, les nouveaux sapins de taille constante sont dessinés avec une guirlande formée de cercles de différentes couleurs.
- Si la case n'est pas cochée, les nouveaux sapins sont également dessinés mais avec une taille variable.

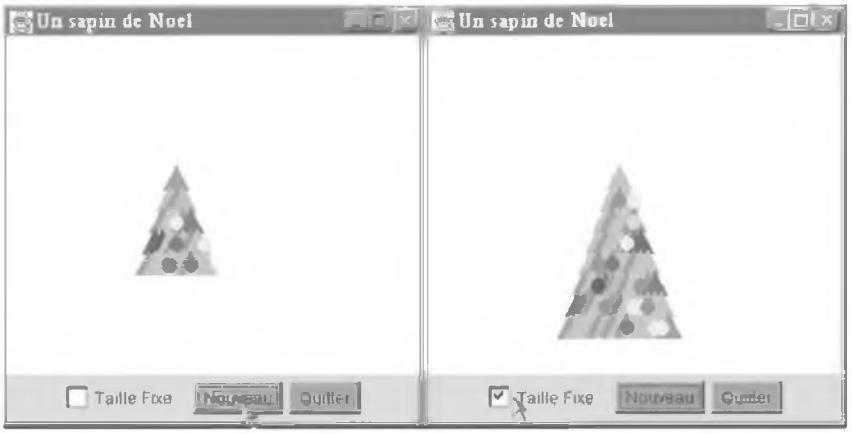


Figure 11-9 *L'application propose une case à cocher pour déterminer si la taille des nouveaux sapins doit être fixe ou non.*

Pour construire cette application, nous vous proposons de suivre les différentes étapes décrites ci-dessous.

Comprendre les techniques d'affichage graphique

Exercice 11.1 Pour afficher un sapin de taille différente chaque fois que l'utilisateur clique sur le bouton `Nouveau` :

a. Recherchez dans l'ensemble des classes de l'application `Fenetre`, la méthode associée au clic sur le bouton `Nouveau`.

b. Modifiez cette méthode de façon à ce que l'arbre se construise avec une taille aléatoire, variant entre `3` et `10`, par exemple.

c. Une fois ces modifications réalisées, compilez l'application, et vérifiez le bon fonctionnement du bouton `Nouveau`.

Copyright © 2016 Eyrolles.

Exercice 11.2 Pour dessiner une guirlande de cercles de couleurs différentes :

a. Avant d'afficher une guirlande, modifiez les classes `Triangle` et `Arbre` de sorte que le sapin ne soit affiché qu'à l'aide de triangles verts. Vérifiez l'exécution du programme.

b. Pour afficher une guirlande de couleur rouge, créez une classe `Boule` en vous inspirant de la classe `Triangle`.

c. Modifiez ensuite la méthode `dessine()` de la classe `Arbre`, de façon à construire et à afficher par-dessus le sapin des objets `Boule` lorsque le tableau `sapin` vaut `1`.

Compilez et exécutez le programme afin de vérifier le bon affichage de la guirlande.

d. Pour afficher une guirlande de couleurs différentes, définissez dans la classe `Boule` un tableau de plusieurs couleurs, comme suit :

```
Color [] couleur = {Color.red, Color.blue, Color.yellow,
                    Color.cyan, Color.magenta};
```

Le choix de la couleur est ensuite effectué dans le constructeur de la classe `Boule` en tirant au hasard une valeur comprise entre `0` et `4`. Cette valeur est utilisée comme indice du tableau de couleurs pour initialiser la couleur d'affichage (`setColor()`) à la couleur du tableau correspondant à l'indice tiré au hasard.

Remarque Notez que la méthode `fillOval(x, y, l, h)` permet l'affichage de cercles remplis. Elle s'applique à un objet `Graphics`, comme la méthode `fillPolygon()`. Les paramètres `x` et `y` représentent la position à l'écran du coin supérieur gauche du rectangle englobant le cercle, `l` et `h` représentant la largeur et la hauteur de ce même rectangle.

Apprendre à gérer les événements

Exercice 11.3 Placer une case à cocher dans la boîte à boutons :

a. Sachant que la classe décrivant les cases à cocher a pour nom `Checkbox`, ajoutez un objet de ce type dans la boîte à boutons de l'application `Fenetre`. Le texte (`" Taille fixe "`) suivant la case à cocher est placé en paramètre du constructeur de la classe.

b. L'écouteur d'événement associé aux objets de type `Checkbox` s'appelant `ItemListener`, ajoutez cet écouteur à la case à cocher.

Exercice 11.4 Associer l'événement à l'action. Lorsque la case est cochée, les nouveaux sapins affichés par le bouton `Nouveau` sont de taille fixe. Inversement, lorsque la case n'est pas cochée, les sapins sont de taille aléatoire. L'état de la case à cocher a donc une influence sur l'affichage du sapin géré par le bouton `Nouveau`. C'est pourquoi il est logique de gérer l'écouteur `ItemListener` dans la même classe qu'`ActionListener`.

Copyright © 2016 Eyrolles.

a. Sachant que l'interface `ItemListener` ne définit qu'un seul comportement `itemStateChanged()`, modifiez l'en-tête de la classe `GestionAction` de façon à ce qu'elle implémente les deux interfaces `ActionListener` et `ItemListener` en séparant les deux termes par une virgule.

b. Analysez la méthode `itemStateChanged()` décrite ci-dessous, et déterminez comment déclarer la variable `OK` pour qu'elle puisse être également visible de l'objet `bNouveau`.

```
public void itemStateChanged(ItemEvent e) {
  if(e.getStateChange() == ItemEvent.SELECTED)
    OK = false;
  else OK = true;
 }
```

c. Sachant que les sapins de taille aléatoire sont affichés par l'intermédiaire de la méthode `nouveau()` (classe `Dessin`), modifiez la méthode de façon que la taille du sapin soit fixe ou aléatoire, en fonction de la valeur de la variable `OK`.

Exercice 11.5

L'objectif de cet exercice est de réaliser une application qui :

- affiche des formes géométriques lues à partir du fichier `Formes.txt` créé au cours de l'exercice 10.5 du chapitre précédent ;
- propose à l'utilisateur deux boutons interactifs placés en bas de la fenêtre, comme le montre la figure suivante :

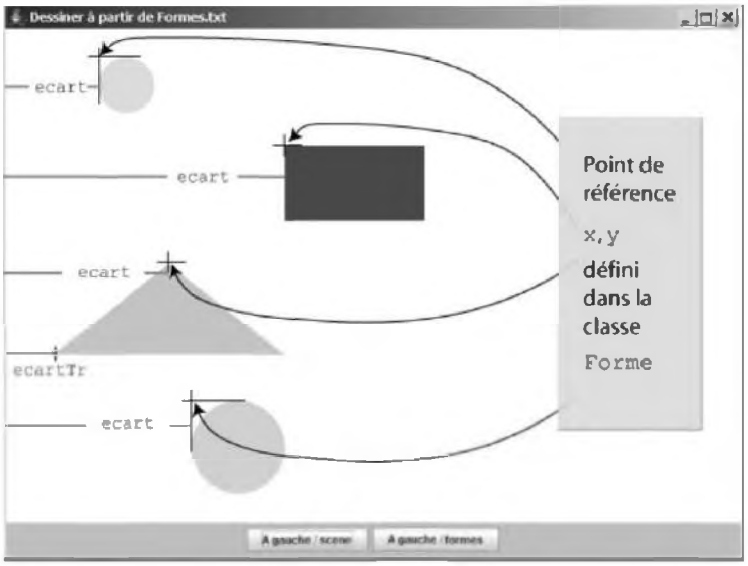


Figure 11-10 *L'application affiche les formes enregistrées dans le fichier Formes.txt ainsi que deux boutons « À gauche / scène » et « À gauche / formes ».*

Copyright © 2016 Eyrolles.

Lorsque l'utilisateur clique sur le bouton « À gauche / scène, les formes géométriques se déplacent toutes sur le bord gauche de la scène, comme le montre la figure 11.11.

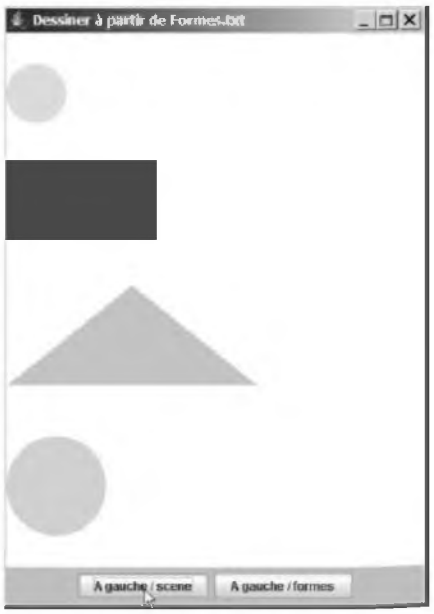


Figure 11-11 *Lorsque l'utilisateur clique sur le bouton « À gauche / scène », les formes se déplacent sur le bord gauche de la fenêtre.*

Lorsque l'utilisateur clique sur le bouton « À gauche / formes », les formes géométriques s'alignent sur la forme la plus à gauche de la scène, comme le montre la figure 11.12.

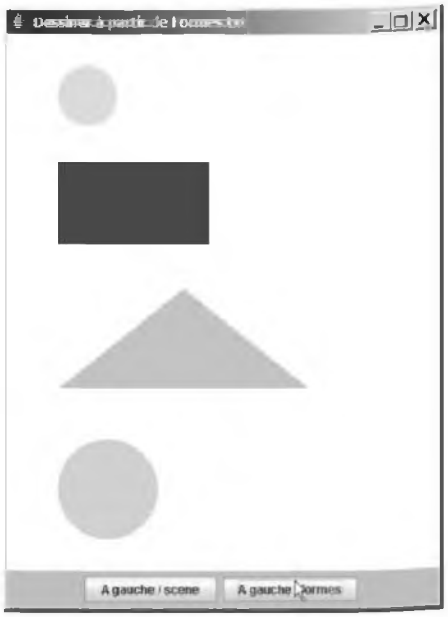


Figure 11-12 *Lorsque l'utilisateur clique sur le bouton « À gauche / formes », les formes s'alignent sur le triangle qui se situe le plus à gauche de la scène.*

Copyright © 2016 Eyrolles.

Extension Web Pour vous faciliter la tâche, vous trouverez dans le répertoire `Source/Exercices/Chapitre11/SupportPourRealiserLesExercices` sur l'extension Web de l'ouvrage, tous les fichiers nécessaires à la réalisation de cette application.

a. Examinez l'ensemble des fichiers contenus dans le répertoire `Source/Exercices/Chapitre11/SupportPourRealiserLesExercices` et déterminez les classes que vous aurez à modifier ainsi que le rôle de chacune d'entre-elles.

b. Dans la classe `ListeDeFormes`, examinez et expliquez ce que réalise la méthode `dessinerLesFormes()`. Quelle est l'instruction qui permet de créer les 2 boutons « À gauche / scène » et « À gauche / formes ».

c. Examinez la classe `DesBoutons` et expliquez comment les boutons sont associés aux actions qu'ils doivent réaliser.

d. Si l'on suppose que le déplacement des formes s'effectue dans la classe `DessinFormes`, à l'aide de deux méthodes nommées `deplacerGaucheScene()` et `deplacerGaucheFormes()`, quelles instructions placeriez-vous dans la classe `GestionAction` pour que les boutons réalisent les actions demandées.

e. Pour déplacer les formes géométriques sur le bord gauche de la fenêtre, la technique consiste à calculer la distance (`ecart`) qui sépare la coordonnée `x` de la forme avec le bord gauche de la scène et de déplacer la forme géométrique de `-ecart` en utilisant la méthode `deplacer()` propre à la forme (voir figure 11-10).

   Dans la classe `DessinFormes`, écrire la méthode `deplacerGaucheScene()` en utilisant cette technique.

   Testez le bouton « À gauche / scène » et vérifiez que toutes les formes se déplacent correctement. Que se passe-t-il pour les formes triangulaires ? Pourquoi ?

   Pour déplacer un triangle,

   Ecrire, dans la classe `Triangle`, une méthode recherchant le sommet se situant le plus à gauche.

   Calculer alors la distance (`ecartTr`) qui sépare la coordonnée `x` de ce sommet avec le bord gauche de la scène

   Déplacer le triangle de `-ecartTr` en utilisant la méthode `deplacer()` de la classe `Triangle`.

g. Pour aligner les formes sur la forme géométrique se situant le plus à gauche, vous devez :

   Rechercher le point le plus à gauche pour toutes les formes, y compris pour le triangle – c'est-à-dire rechercher la plus petite coordonnée en `x`, parmi toutes les coordonnées en `x`.

   Pour chaque forme, calculez la distance entre son point de référence et la plus petite coordonnée en `x` et déplacez la forme de cette distance.

   Dans la classe `DessinFormes`, écrire la méthode `deplacerGaucheForme()` en utilisant cette marche à suivre.

Copyright © 2016 Eyrolles.

Le projet : Gestion d'un compte bancaire

L'objectif est de réaliser des statistiques sur les comptes bancaires enregistrés dans les fichiers créés au chapitre précédent. Le résultat de ces statistiques est affiché dans une fenêtre, comme illustré à la figure 11-13.

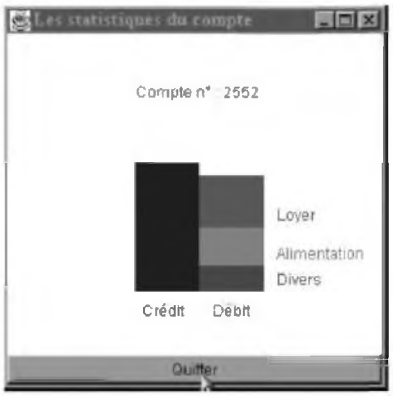


Figure 11-13 *Histogramme empilé du compte n° 2552*

Calcul de statistiques

Les classes ListeCompte et Compte

En reprenant la fonction `pourcentage()` réalisée en exercice à la fin du chapitre 5, « De l'algorithme paramétré à l'écriture de fonctions », et sachant que l'objectif est de calculer en pourcentage les dépenses réalisées en fonction du motif de la dépense :

a. Déterminer toutes les données, définies dans les classes `Compte` et `LigneComptable`, nécessaires aux calculs des statistiques d'un compte.

b. Vérifier que ces données soient accessibles depuis l'extérieur de la classe. Si tel n'est pas le cas, écrire les méthodes d'accès en consultation pour chacune d'entre elles.

c. Après modifications, compiler et exécuter le programme de façon à créer un fichier de comptes (`Compte.dat`).

La méthode statParMotif()

a. En reprenant l'algorithme de calcul de statistiques proposé en exemple du chapitre 1, « Stocker une information », écrire la méthode `statParMotif()` qui calcule le pourcentage des dépenses en fonction du motif enregistré.

b. Sachant que cette méthode est définie à l'intérieur d'une classe appelée `Stat`, déterminer les variables d'instance de cette classe.

Copyright © 2016 Eyrolles.

c. Avant de passer à l'affichage graphique, écrire une application qui :
   - lise le fichier `Compte.dat` créé à l'étape précédente ;
   - utilise la méthode `statParMotif()` pour calculer et afficher à l'écran les statistiques d'un compte donné.

d. Vérifier la validité des calculs réalisés.

L'interface graphique

Pour calculer les statistiques d'un compte, l'utilisateur doit fournir le numéro du compte choisi. Après lecture du fichier `Compte.dat`, et connaissant ce numéro, l'application vérifie s'il existe en mémoire. Si tel est le cas, elle affiche dans une fenêtre le résultat sous forme d'histogrammes empilés. Dans le cas contraire, elle affiche un message indiquant que ce compte n'est pas connu et attend la saisie d'un nouveau numéro.

Deux étapes sont donc à réaliser, la saisie d'un numéro de compte et l'affichage de l'histogramme.

L'affichage de l'histogramme

Pour afficher l'histogramme empilé, il est nécessaire de connaître les pourcentages de dépenses en fonction des motifs déclarés. Ces valeurs sont calculées dans la classe `Stat`, construite précédemment.

a. En s'inspirant de la méthode `dessine()` présentée en exemple au cours de ce chapitre, écrire dans la classe `Stat` la méthode `dessine()` de façon à afficher :
   - un premier rectangle de hauteur `100` et de largeur `50` représentant l'unité de crédit (`100`) ;
   - des rectangles de couleur et de hauteur différentes suivant les pourcentages calculés par la méthode `statParMotif()`.

   Noter que l'affichage d'un rectangle rempli s'effectue par l'intermédiaire de la méthode `fillRect(x, y, l, h)`, où `x` et `y` représentent la position à l'écran du coin supérieur gauche du rectangle, et `l` et `h` sa largeur et sa hauteur. L'affichage d'un texte est réalisé par la méthode `drawString(texte, x, y)`, où `texte` est un objet de type `String` dans lequel sont placés les caractères à afficher, `x` et `y` définissant la position de ce texte à l'écran.

b. Définir une fenêtre composée d'une zone de dessin et d'un bouton `Quitter`.

c. L'affichage de l'histogramme étant réalisé dans la zone de dessin,
   - Le constructeur de la fenêtre doit prendre en paramètre un objet de type `Stat` de façon à le transmettre au constructeur de la zone de dessin.
   - La méthode `paint()` définie dans la classe représentant la zone de dessin fait appel à la méthode `s.dessine()`, où `s` est un objet de type `Stat`, initialisé dans le constructeur de la zone de dessin.

Copyright © 2016 Eyrolles.

d. Le bouton `Quitter` et l'icône de fermeture située dans la barre de titre de la fenêtre ayant la même fonctionnalité (quitter l'application et fermer la fenêtre) :

- Créer une classe `GestionQuitter` qui implémente l'écouteur `ActionListener` et dérive de la classe `WindowAdapter`.
- Définir les méthodes correspondant au comportement de fermeture d'application.

La saisie d'un numéro de compte

La classe `Saisie` décrite ci-dessous permet la saisie d'une chaîne de caractères par l'intermédiaire d'une fenêtre de saisie :

```
import java.awt.*;
import java.awt.event.*;
public class Saisie implements ActionListener {
  TextField réponse;
  public Saisie ()  {
    Frame F = new Frame ("Saisie de valeurs:");
    F.setSize(300, 50);
    F.setBackground(Color.white);
    F.setLayout(new BorderLayout());
    F.add (new Label("Valeur :"), "West");
    réponse = new TextField(10);
    F.add(réponse, "East");
    réponse.addActionListener(this);
    F.setVisible(true);
  }
  public void actionPerformed(ActionEvent evt) {
    String numéro = réponse.getText();
    System.out.println(numéro);
  }
}
```

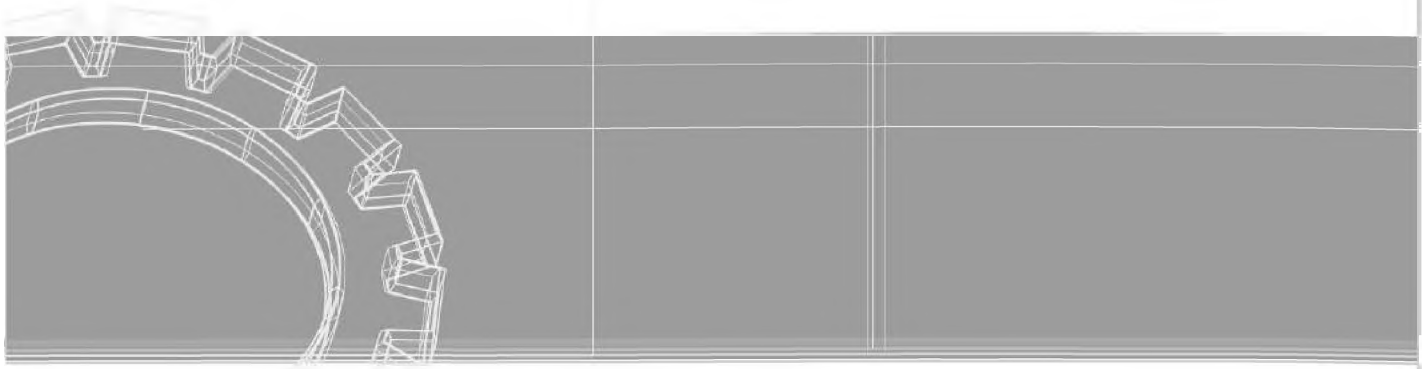
Observer et analyser cette classe et transformer la méthode `actionPerformed()` de façon à calculer puis à afficher l'histogramme cumulé si le numéro de compte lu dans la fenêtre de saisie correspond à un compte enregistré dans le fichier `Compte.dat`.

Copyright © 2016 Eyrolles.

Copyright © 2016 Eyrolles.

Chapitre 12

Créer une interface graphique



Nous l'avons vu au cours du chapitre précédent, créer des fenêtres, dessiner des objets et définir des interactions entre tous ces éléments demande un certain savoir-faire et une bonne connaissance des bibliothèques graphiques comme l'AWT, Swing ou encore SWT.

Pour simplifier le développement d'applications graphiques, des outils d'aide à la création d'interfaces graphiques ont été développés par des sociétés telles que Sun ou IBM.

Nous présentons, à la section « Un outil d'aide à la création d'interfaces graphiques », l'application NetBeans utilisée pour développer des interfaces Java. Il s'agit d'un environnement de développement distribué sous licence Open Source.

Puis, au cours des deux sections suivantes, « Gestion de bulletins de notes » et « Un éditeur pour dessiner », nous présentons pas à pas comment utiliser ce logiciel pour construire des interfaces graphiques. La première application a pour objectif de faciliter la création et l'édition de bulletins de notes. La seconde application, plus classique, montre comment construire les bases d'un éditeur graphique.

Un outil d'aide à la création d'interfaces graphiques

Les outils d'aide à la création d'interfaces graphiques sont appelés dans le monde informatique, des IDE (*Integrated Development Environment*) que l'on traduit en français par Environnement de développement intégré (EDI).

Copyright © 2016 Eyrolles.

Qu'est qu'un EDI ?

Un environnement de développement intégré est un programme ou encore une application qui propose, dans un même système de fenêtrage (environnement), des outils facilitant la vie du développeur. Ces outils sont, au minimum :

- un éditeur de texte qui colorie automatiquement les mots-clés et les éléments syntaxiques importants du langage ;
- un système automatique de correction syntaxique des erreurs les plus communes (auto-complétion) ;
- une fenêtre de compilation où s'affichent les éventuelles erreurs de compilation ;
- une fenêtre d'exécution qui permet de visualiser les résultats de l'application en cours de développement.

Certains EDI sont dédiés à un seul langage (Visual Basic, Delphi, etc.), d'autres supportent plusieurs langages, comme Eclipse avec lequel il est possible de développer en Java, C++, ActionScript 3, etc.

Les EDI proposent également des outils graphiques très performants qui permettent de construire graphiquement l'interface d'une application en plaçant, à la souris, les éléments de communication (fenêtre de saisie, bouton de validation, zone de dessin, liste de choix, etc.).

Pour le langage Java, il existe plusieurs EDI, dont :

- JBuilder (Bordland) ;
- JCreator (Xinox Software) ;
- Eclipse (Eclipse Foundation - IBM) ;
- NetBeans (Sun).

NetBeans a l'avantage d'être distribué en Open Source et d'être entièrement gratuit. Pour cette raison et pour sa facilité d'installation et d'usage, nous avons choisi de vous le présenter en détail.

Les bases de NetBeans

NetBeans est un environnement de développement intégré pour Java, créé par la société Sun. Il supporte plusieurs langages de programmation comme C, C++, Java ou encore Python.

La dernière version NetBeans 8.1 présentée ici est disponible sur tout type de plate-forme (Windows, Linux et Mac).

Pour en savoir plus Toutes les informations nécessaires à l'installation de NetBeans sur votre machine (Windows, Mac OS ou Linux) sont fournies dans l'annexe « Guide d'installations », section « Installation d'un environnement de développement ».

Copyright © 2016 Eyrolles.

Présentation générale

NetBeans est constitué d'une fenêtre d'édition du code avec coloration syntaxique et surtout d'un éditeur graphique d'interfaces et de pages Web (voir figure 12-1). Grâce à l'éditeur graphique, le développeur d'applications dispose visuellement, d'une boîte à outils (palette) constitué des éléments graphiques de la bibliothèque Swing.

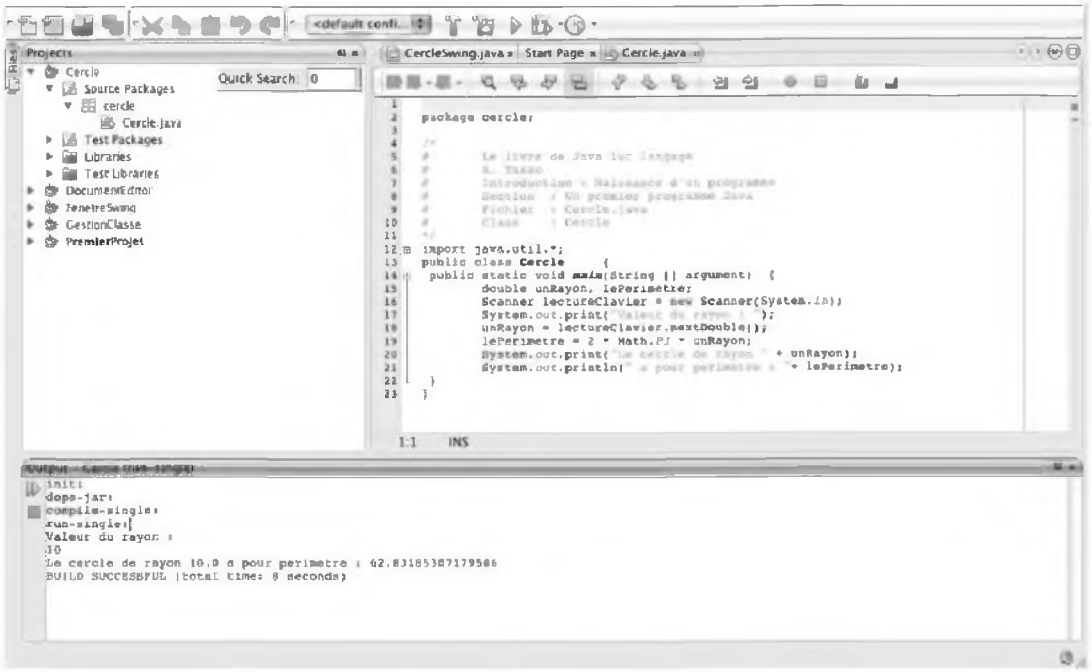


Figure 12-1 *NetBeans 8.1*

Pour en savoir plus Les principaux éléments de la bibliothèque Swing sont présentés au chapitre 11, « Dessiner des objets », section « De l'AWT à Swing ».

L'éditeur graphique de NetBeans est un éditeur WYSIWYG (*What You See Is What You Get*), ce qui signifie en français que ce que l'on place (*What You See*) dans la fenêtre d'édition graphique est ce que l'on obtient (*Is What You Get*) en cours d'exécution de l'application.

Grâce à cela, il devient très facile d'intégrer graphiquement les éléments d'interactions avec l'utilisateur (boîte de dialogue, cases à cocher, menu, etc.) dans l'application en cours de développement. Cette intégration s'effectue par un simple glisser-déposer des éléments de la bibliothèque vers la fenêtre de développement.

Développer une interface graphique en mode projet

Sous NetBeans, la création et l'exécution d'une application s'effectuent dans le cadre d'un projet au sein duquel sont regroupées toutes les classes nécessaires à la bonne marche de l'application.

Copyright © 2016 Eyrolles.

Pour créer un projet vous devez, une fois NetBeans lancé, sélectionner l'item Nouveau Projet du menu Fichier.

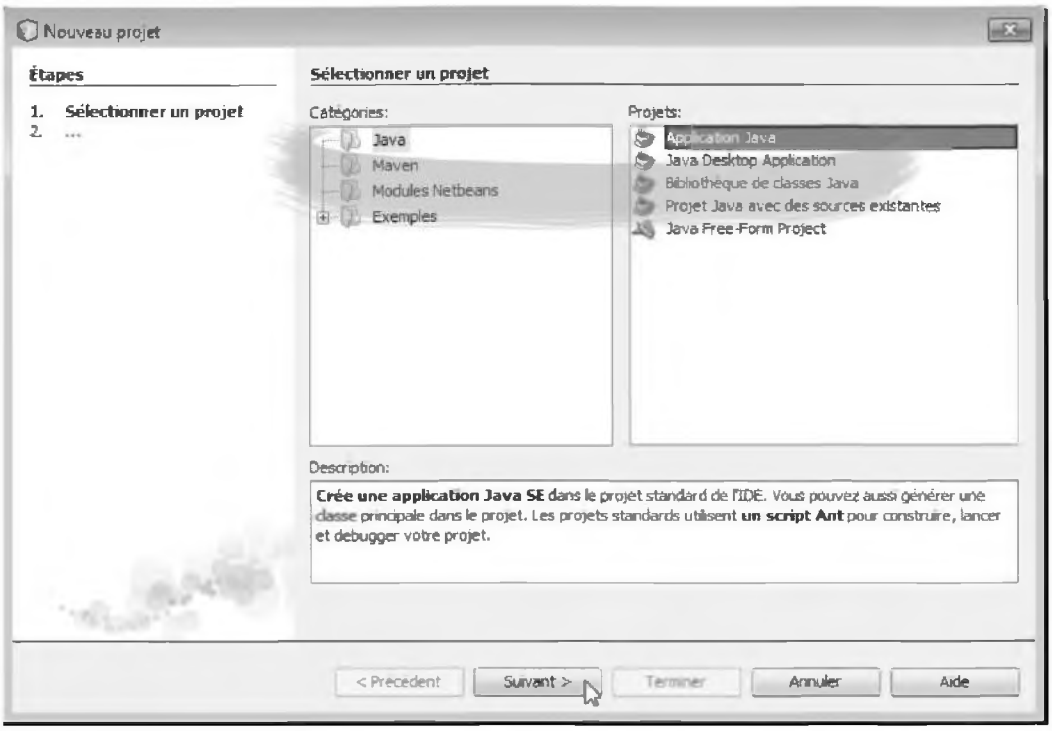


Figure 12-2 *La boîte de dialogue Nouveau Projet*

Dans la boîte de dialogue qui apparaît (voir figure 12-2), choisir la catégorie Java et comme type de projet Application Java. Cliquez ensuite sur le bouton Suivant.

Une nouvelle boîte de dialogue apparaît, nommée Nouveau Application Java (voir figure 12-3). Ici, vous devez :

- Entrer le nom du projet en première ligne.
- Indiquer le lieu d'enregistrement du projet en cliquant sur le bouton Parcourir..., ou garder les valeurs par défaut proposées par NetBeans.
- Désélectionner la case Créer une classe principale. Pour les applications de type « interface graphique », le point d'entrée de l'exécution de l'application est défini à l'étape suivante.
- Finir l'enregistrement du projet, en cliquant sur le bouton Terminer.

Une fois le projet créé et enregistré, NetBeans affiche à l'écran (voir figure 12-4) un ensemble de panneaux vides, à l'exception du panneau Projets qui indique que le projet `PremierProjet` a bien été créé.

Copyright © 2016 Eyrolles.

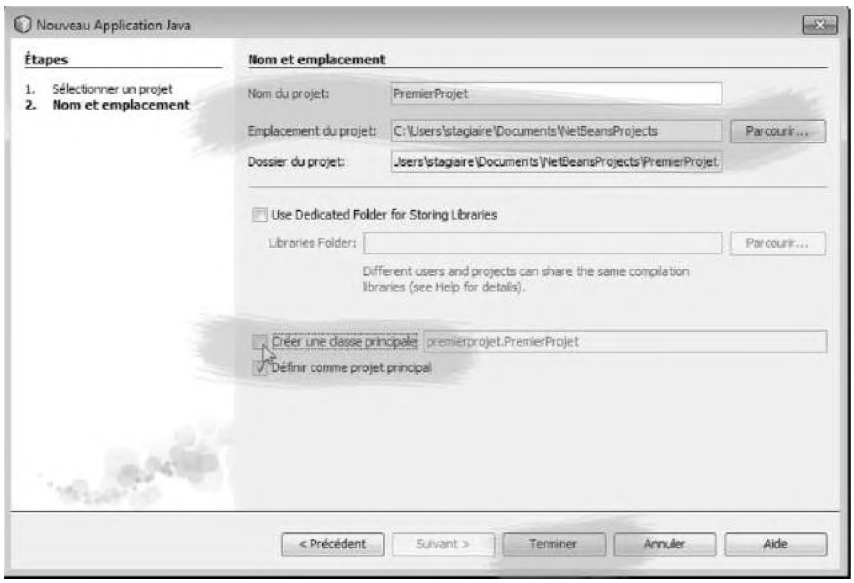


Figure 12-3 *La boîte de dialogue Nouveau Application Java. Attention à bien désélectionner la case Créer une classe principale.*

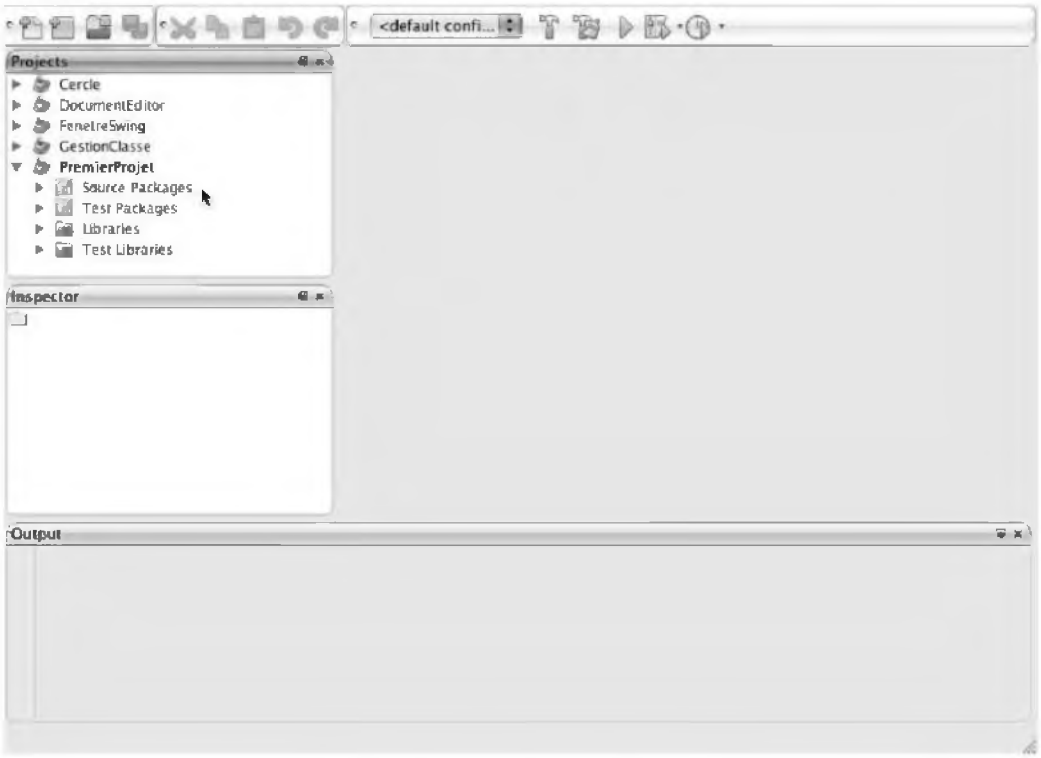


Figure 12-4 *Le panneau Projets*

Copyright © 2016 Eyrolles.

Développer en mode graphique

La dernière étape pour construire une application à l'aide de l'éditeur graphique consiste à ajouter au projet une fenêtre de type `JFrame`. Pour cela :

- cliquer droit sur l'item PremierProjet situé dans le panneau Projets ;
- sélectionner les items Nouveau puis JFrame Form comme le montre la figure 12-5.

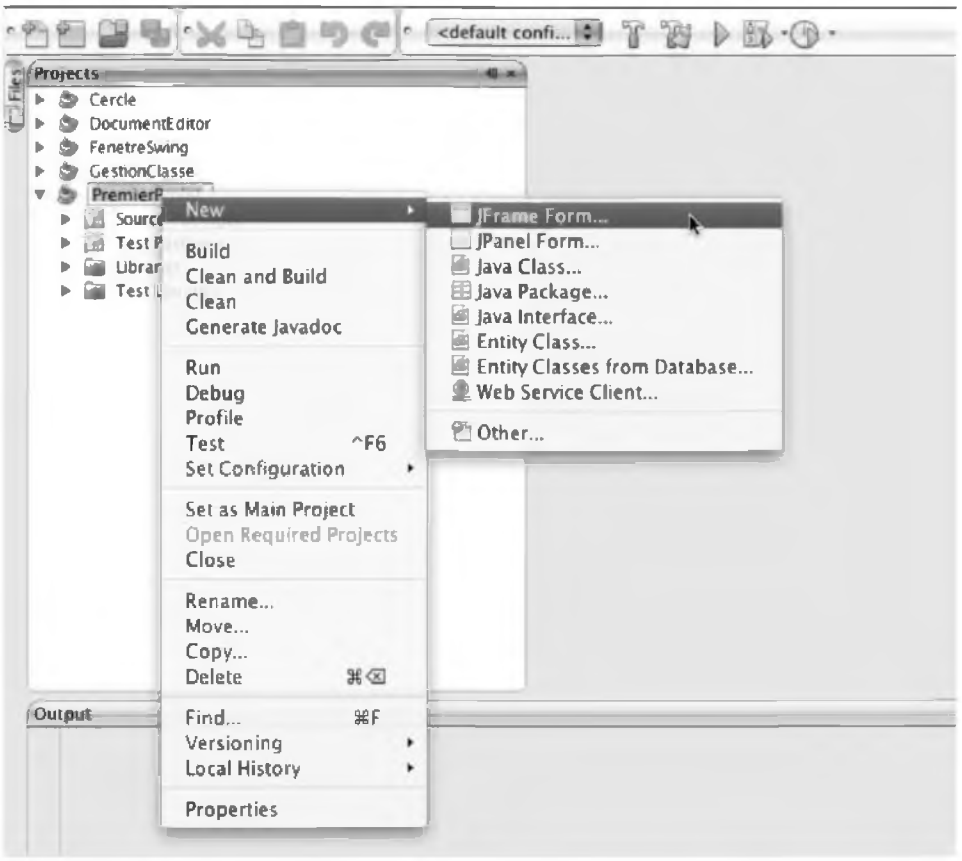


Figure 12-5 *Créer une fenêtre graphique*

Pour en savoir plus Les objets de type `JFrame` sont présentés au chapitre 11, « Dessiner des objets » section « De l'AWT à Swing ».

La boîte de dialogue Nouveau JFrame Form apparaît (voir figure 12-6). Il convient d'indiquer ici le nom de classe dans le champ Class Name ainsi que le nom du package au sein duquel vous souhaitez enregistrer les classes utilisées par l'application en cours de construction.

Copyright © 2016 Eyrolles.

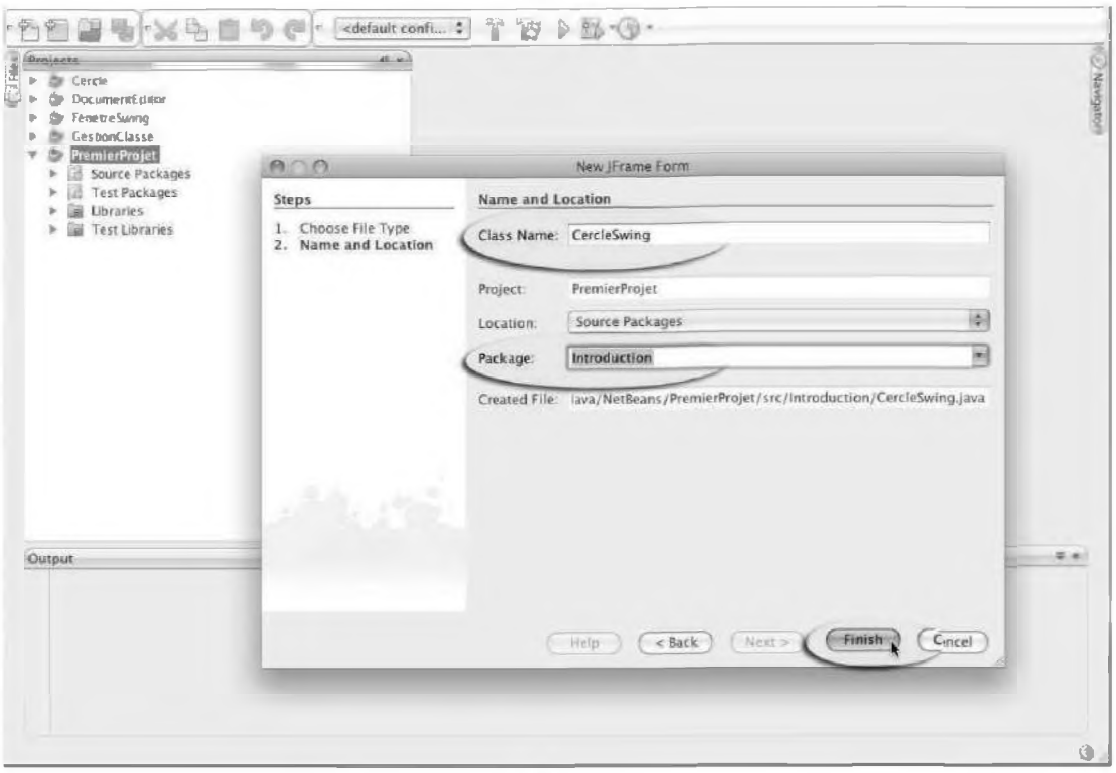


Figure 12-6 *La boîte de dialogue Nouveau JFrame Form*

Pour notre exemple, nous avons choisi `CercleSwing` comme nom de classe et `Introduction` comme nom de package.

Remarque

Définir un package pour notre application revient à définir le point d'entrée d'une arborescence de répertoires au sein de laquelle sont stockées les classes utilisées par l'application. Pour notre exemple, le point d'entrée pour l'enregistrement des classes Java est le répertoire `PremierProjet/src/Introduction`.

Cela fait, nous obtenons un environnement de travail composé de plusieurs panneaux ayant chacun un rôle particulier (voir figure 12-7). Nous présentons ci-après, les panneaux les plus utiles pour concevoir des interfaces graphiques.

Pour en savoir plus

L'organisation du système de fichiers créé par NetBeans est décrite dans l'annexe « Guide d'installations », section « Utilisation des outils de développement ».

Copyright © 2016 Eyrolles.

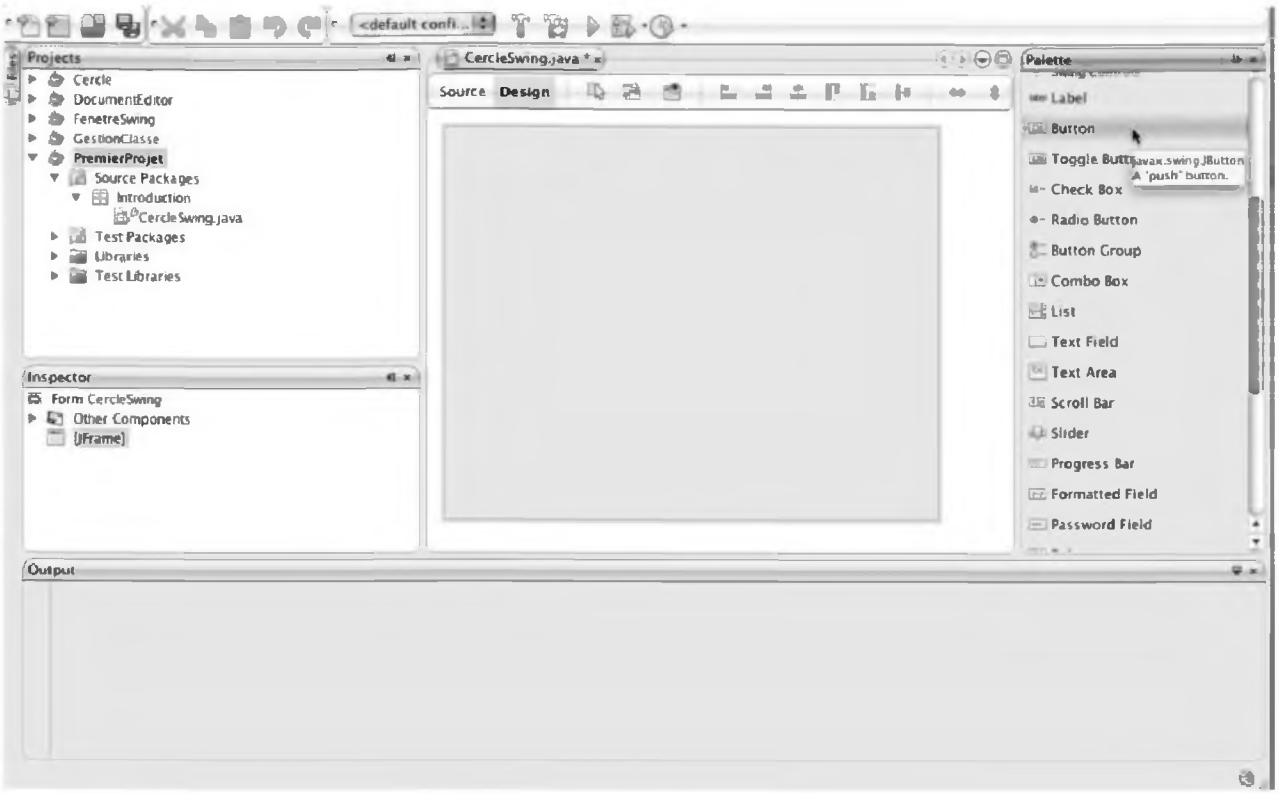


Figure 12-7 *L'éditeur graphique de NetBeans*

Le panneau Design

Le panneau central appelé panneau Design est le panneau principal pour concevoir une application graphique (voir figure 12-8). Il représente le fond de votre application, votre page blanche sur laquelle vous allez placer tous les éléments d'interactions (boutons, zone de texte, ...). Grâce à ce panneau, vous allez concevoir visuellement l'interface graphique de votre application.

Le panneau Design peut être vu d'une façon plus « écrite » en cliquant sur l'onglet Source situé à gauche de l'onglet Design.

Le panneau Source

Le panneau Source (voir figure 12-9) est la représentation programmée du panneau Design. On peut y voir toutes les instructions nécessaires à l'affichage de l'interface graphique.

Remarque

Pour passer d'une représentation graphique à une représentation « codée », il suffit de cliquer sur l'onglet correspondant (Design ou Source).

Copyright © 2016 Eyrolles.

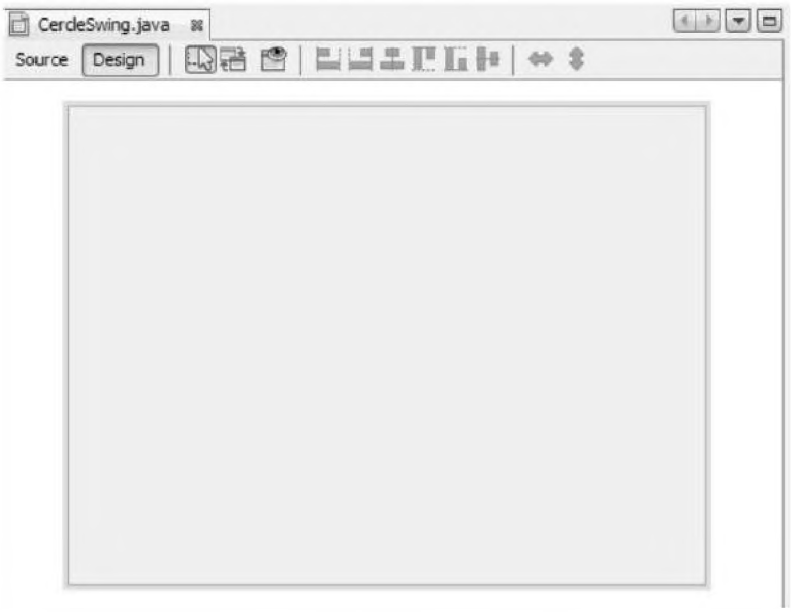


Figure 12-8 *Le panneau Design (Conception)*

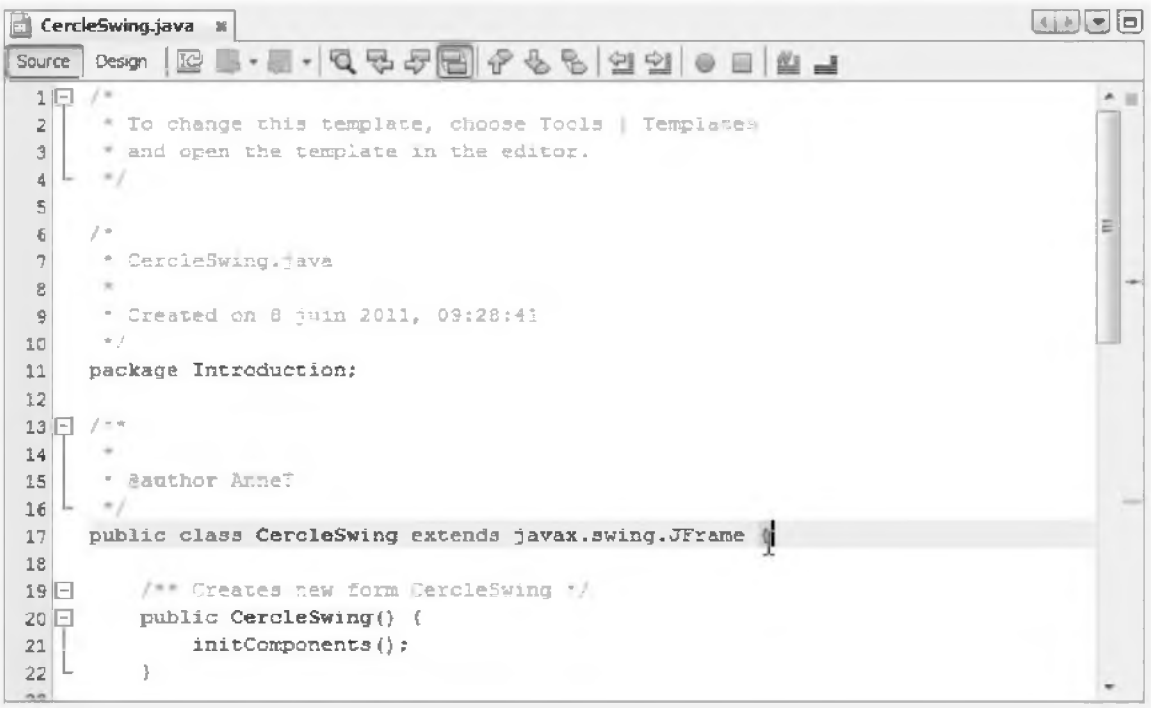


Figure 12-9 *Le panneau Source*

Copyright © 2016 Eyrolles.

Le code source de l'application construite visuellement par vos soins est autogénéré par NetBeans. La plus grande partie des instructions est masquée afin de rendre la lecture du code plus aisée. Il est cependant possible de le visualiser en entier en cliquant sur les signes + situés en marge gauche du code.

Le panneau Palette

Le panneau Palette (voir figure 12-10) est très utile puisqu'il contient tous les composants proposés par la bibliothèque Swing, comme les composants `JButton` et `JPanel` que nous avons étudiés au cours du chapitre précédent « Dessiner des objets », section « De l'AWT à Swing ».

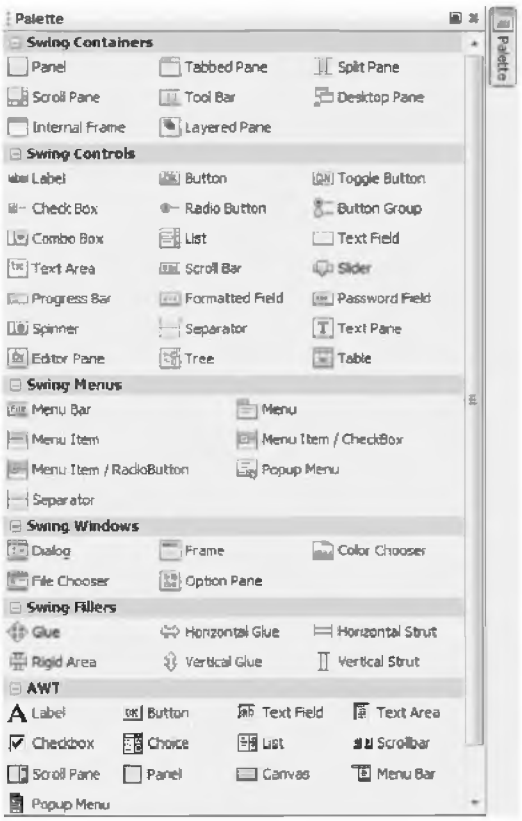


Figure 12-10 *Le panneau Palette*

Pour placer un composant sur le panneau Design, il suffit de le sélectionner dans le panneau Palette puis de le glisser vers le panneau Design. Le positionnement du composant au sein de la fenêtre s'effectue ensuite visuellement, en le déplaçant avec la souris.

Le panneau Propriétés

Le panneau Propriétés (voir figure 12-11) est un panneau contextuel qui permet de visualiser l'intégralité des propriétés du composant sélectionné. Par exemple, sur la figure 12-11, nous pouvons voir les propriétés du fond (`JFrame`) de la fenêtre de l'application en cours de construction.

Copyright © 2016 Eyrolles.

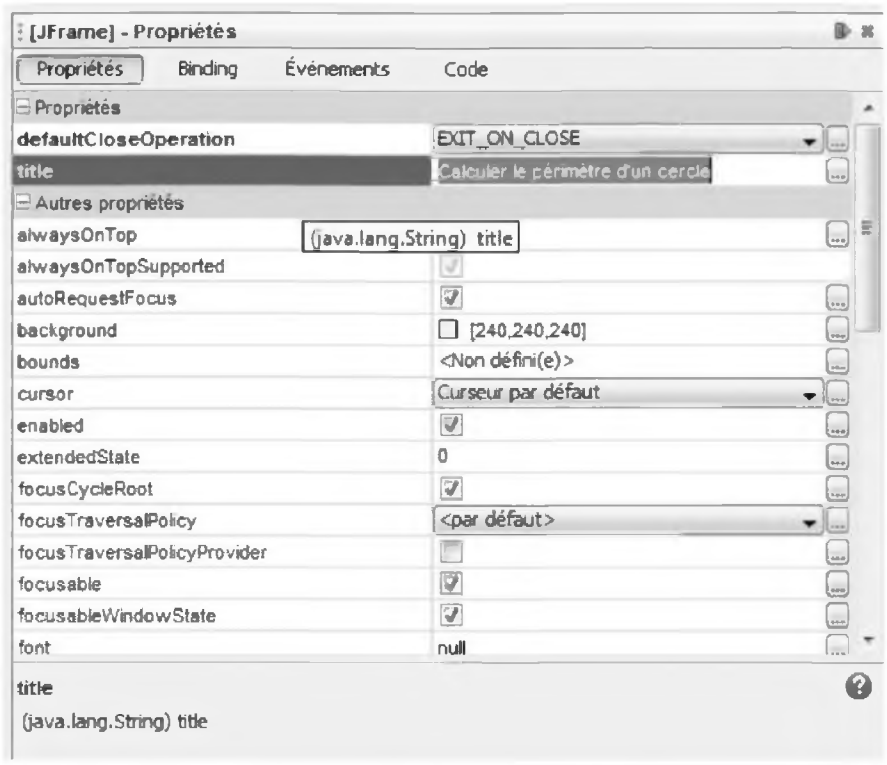


Figure 12-11 *Le panneau Propriétés*

Pour modifier l'une des propriétés du composant, il suffit de la sélectionner et d'indiquer la nouvelle valeur dans la colonne correspondante. Par exemple, sur la figure 12-11, nous avons donné comme titre à la fenêtre de base de l'application : « Calculer le périmètre d'un cercle ».

Le panneau Inspecteur

Pour finir, avec le panneau Inspecteur (voir figure 12-12) nous visualisons les composants utilisés par l'application. Ici le seul composant utilisé est un `JFrame`.

Figure 12-12 *Le panneau Inspecteur*

Copyright © 2016 Eyrolles.

Grâce au panneau Inspecteur, nous pourrons nommer les composants et modifier leur hiérarchie s'il se trouve qu'un composant contienne d'autres composants.

Vous vous familiariserez rapidement avec l'utilisation de ces panneaux et en cernerez l'importance dès la mise en place du premier exemple présenté à la section suivante, « Une première application avec NetBeans ».

Une première application avec NetBeans

La création d'une application Java munie d'une interface de communication graphique s'effectue en deux étapes.

1. Choisir, placer et nommer à l'écran les objets permettant l'interaction avec l'utilisateur. Il s'agit, par exemple, de définir s'il est préférable de proposer à l'utilisateur un choix d'options sous la forme d'un menu ou d'un ensemble de cases à cocher.
2. Écrire, pour chaque élément d'interaction choisi, les actions à mener. Par exemple, si l'application affiche un bouton Quitter, il convient d'écrire en Java, l'instruction `System.exit(0)` à l'intérieur de la fonction qui traite ce bouton.

Examinons sur un exemple simple et « bien connu », à savoir « calculer le périmètre d'un cercle », comment réaliser chacune de ces étapes.

Cahier des charges

L'objectif est de créer et d'afficher la fenêtre représentée à la figure 12-13.

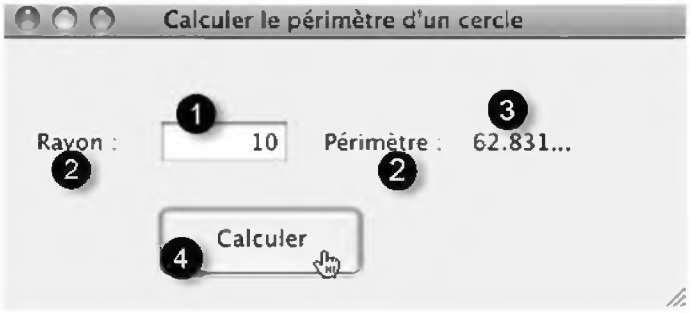


Figure 12-13 *Calculer le périmètre d'un cercle*

La fenêtre est composée de plusieurs éléments graphiques :

- un champ de saisie (voir figure 12-13-❶) qui permet à l'utilisateur de transmettre à l'application la valeur du rayon du cercle dont il souhaite connaître le périmètre ;
- un bouton pour valider la saisie d'une valeur pour le rayon puis, calculer le périmètre du cercle (voir figure 12-13-❹) ;

Copyright © 2016 Eyrolles.

- deux zones de texte (voir figure 12-13-❷) pour placer une information spécifique (« Rayon : » et « Périmètre : »). Ces textes resteront identiques tout au long de l'exécution de l'application ;
- une zone (voir figure 12-13-❸) pour afficher le périmètre du cercle après validation. La valeur sera modifiée à chaque fois que l'utilisateur saisira une nouvelle valeur.

Mise en place des éléments graphiques

La mise en place des composants graphiques dans la fenêtre d'application s'effectue par un simple glisser-déposer des composants du panneau Palette vers le fond de la fenêtre (`JFrame`) visible sur le panneau Design.

Les champs de texte

Dans le panneau Palette, les champs de texte sont représentés par le composant Label (`JLabel`). Les éléments sont placés dans la fenêtre de l'application, comme le montre la figure 12-14-❶.

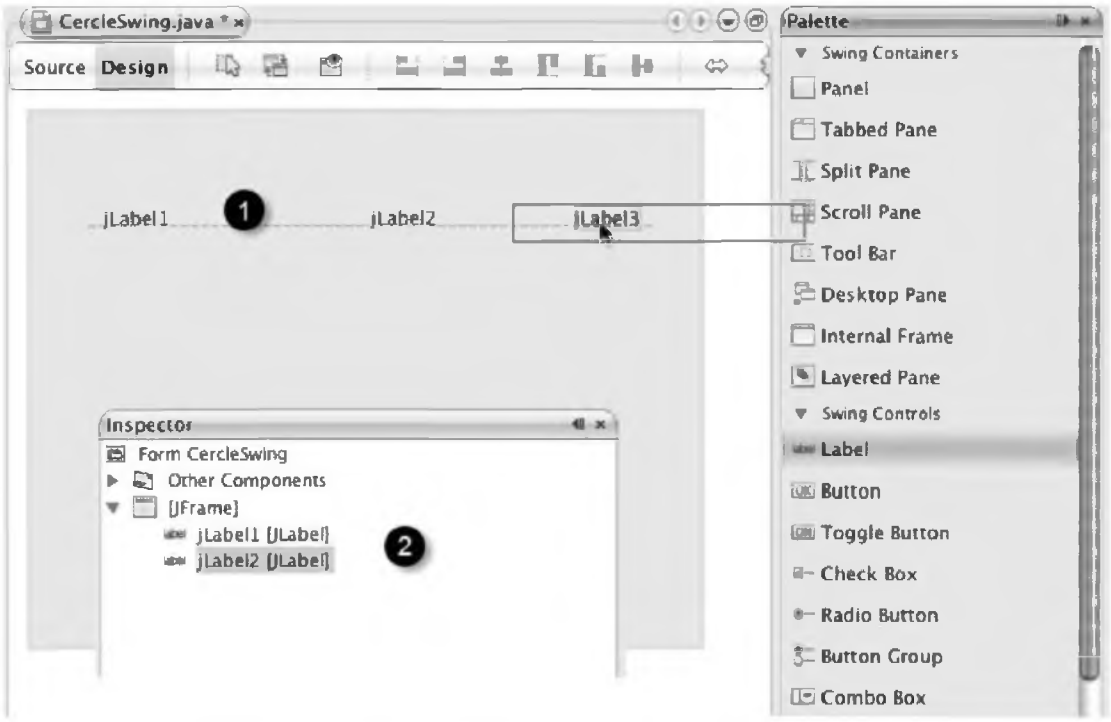


Figure 12-14 *Les champs de texte sont des composants de type Label.*

Observez que pour chaque champ de texte ajouté, le nom du composant ajouté s'affiche dans le panneau Inspecteur (voir figure 12-14-❷).

Copyright © 2016 Eyrolles.

Le nom des objets placés dans la fenêtre porte, par défaut, le nom du composant qu'il représente suivi d'un chiffre qui correspond à son numéro d'ordre de création. Ce nom est porté deux fois par le composant, dans la fenêtre Inspecteur et dans la fenêtre Design :

- Dans le panneau Inspecteur, ce nom représente le nom de l'objet utilisé par le code source autogénéré. Pour rendre ce code plus lisible et faciliter la programmation ultérieure, vous devez modifier ces noms.

  Ainsi, nous appelons `jLabel1`, `rayonLabel`, `jLabel2`, `perimetreLabel`, `jLabel3` et `resultatLabel` les objets utilisés dans ce premier exemple. Pour cela, vous devez, dans le panneau Inspecteur, cliquer deux fois sur chaque élément `jLabel1`, `jLabel2`, etc., en laissant un petit temps d'attente entre les deux clics. Cela fait, le nom du composant sélectionné est surligné en bleu et vous pouvez alors modifier son nom. Le composant change définitivement de nom après validation par la touche Entrée du clavier (voir figure 12-14-❶).

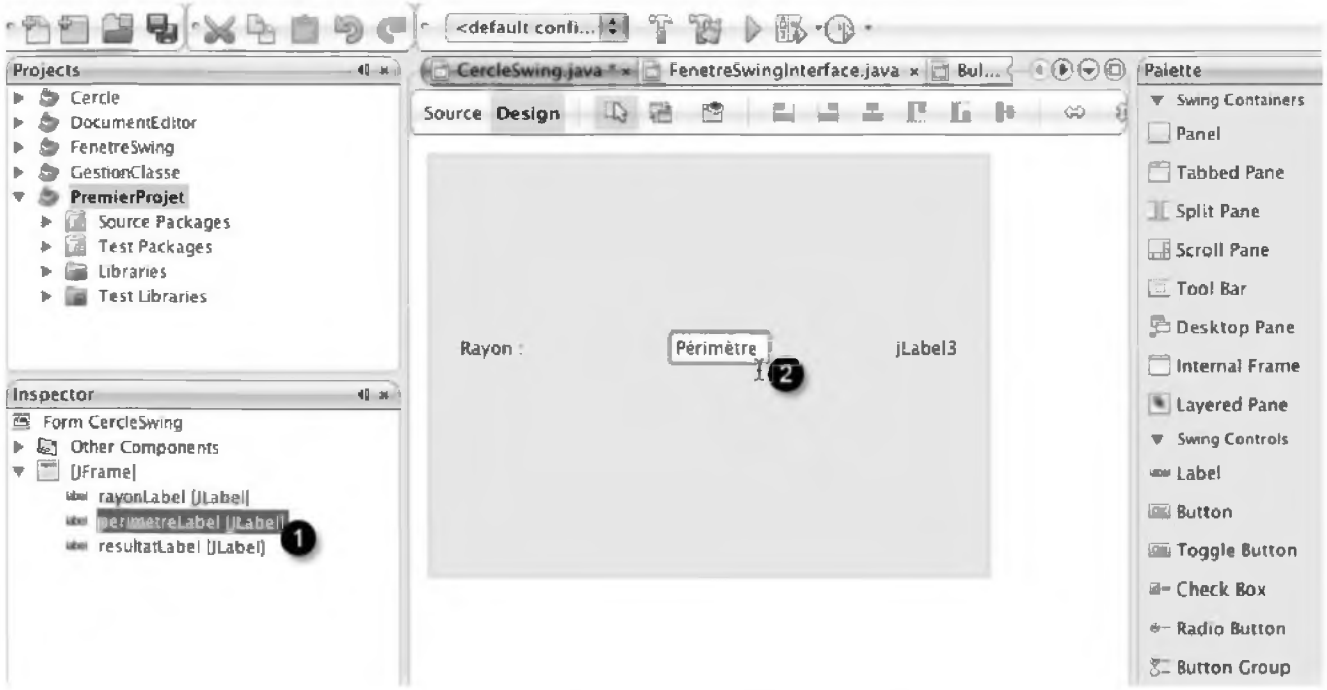


Figure 12-15 *Nommer les composants dans le panneau Inspector (❶) et modifier le texte à afficher dans le panneau Design (❷)*

- Dans le panneau Design, ce nom représente le texte affiché par le composant lors de l'exécution du programme (voir figure 12-15-❷). Vous devez remplacer ces noms par du texte correspondant à ce que vous souhaitez obtenir comme information. Ainsi, tels qu'ils sont situés dans le panneau, le texte `jLabel1` doit être remplacé par `Rayon :` , `jLabel2` par `Périmètre :` et `jLabel3` par `0`.

Copyright © 2016 Eyrolles.

Le champ `resultatLabel` affiche le résultat de l'opération qui consiste à calculer le périmètre d'un cercle. La première fois que l'application est lancée, la valeur du rayon n'est pas encore donnée. C'est pourquoi, nous affichons comme valeur par défaut `0`.

Remarque Lorsque vous modifiez le nom d'un composant par l'intermédiaire du panneau Inspecteur, NetBeans se charge alors de modifier le nom de toutes les occurrences de ce composant qui pourraient se trouver dans le code source associé.

Le champ de saisie

Les champs de saisie sont représentés, dans le panneau Palette, par le composant Text Field (`JTextField`). Le champ de saisie est placé dans la fenêtre de l'application, juste après le champ de texte `rayonLabel`, comme le montre la figure 12-13-❶.

Dans le panneau Inspecteur, nous nommons le champ de saisie `jTextField1`, `saisieRayon` et plaçons comme valeur d'affichage par défaut, la valeur `0`.

Les composants `JTextField` et `JLabel` ont par défaut un alignement horizontal à gauche (texte ferré à gauche). Pour modifier cette propriété, il suffit d'ouvrir le panneau Propriétés en cliquant droit sur le composant à modifier et en sélectionnant l'item Propriétés du menu contextuel qui apparaît alors.

Dans le panneau Propriétés (voir figure 12-16), sélectionnez la propriété horizontalAlignment sur la colonne de gauche et choisissez la valeur Right dans la menu qui apparaît, lorsque l'on clique sur la colonne de droite.

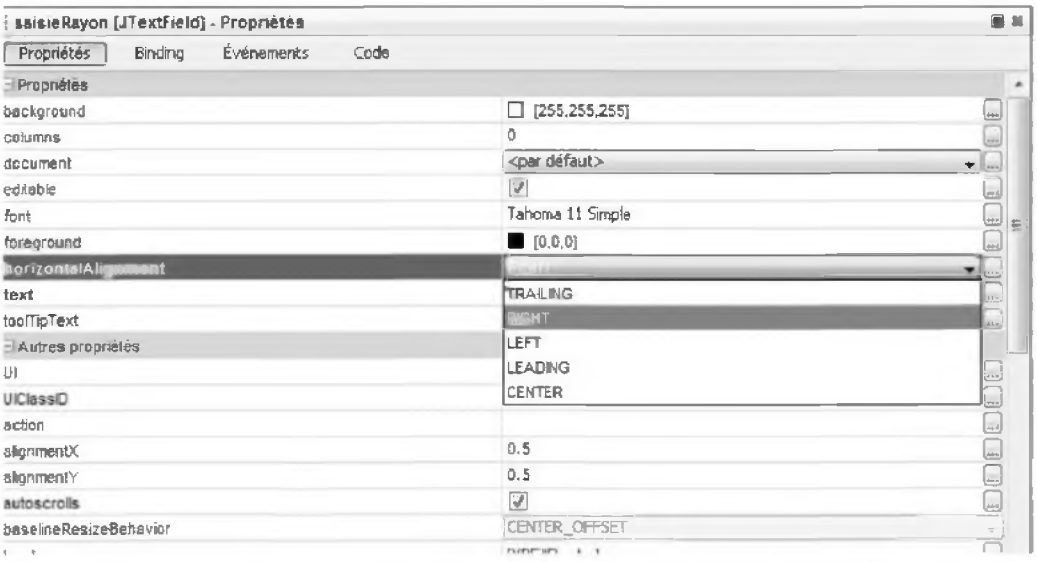


Figure 12-16 *Modifier la propriété horizontalAlignment du champ de saisie saisieRayon*

Copyright © 2016 Eyrolles.

Le bouton

Pour finir, plaçons le bouton de validation dans la fenêtre d'application, comme le montre la figure 12-13-❹.

Les boutons sont représentés, dans le panneau Palette, par le composant Button (`JButton`). Dans le panneau Inspecteur, nous nommons le bouton `jButton1`, `calculerBtn` et plaçons comme texte d'affichage `Calculer`.

Ainsi, lorsque tous les composants sont placés dans la fenêtre d'application, nous obtenons le panneau Inspecteur présenté en figure 12-17.

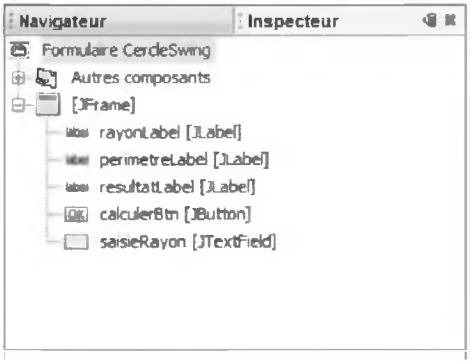


Figure 12-17 *Le panneau Inspecteur de l'application CercleSwing.java*

Ajustement et positionnement des composants

Afin de rendre l'interface plus agréable à l'œil, faisons en sorte que tous les composants soient de même largeur. Pour cela, il suffit de :

- Sélectionner les composants soit dans le panneau Inspecteur, soit directement dans le panneau Design en maintenant la touche Maj enfoncée, à chaque sélection pour réaliser une sélection multiple.
- Cliquer droit sur la sélection et choisir l'item Même Taille puis Même Largeur (voir figure 12-18).

Le menu contextuel apparaissant au clic droit, sur un ou plusieurs composants de l'application, permet de modifier la taille et la position des éléments les uns par rapport aux autres ou par rapport au fond de la fenêtre (item Ancre).

Enfin, pour modifier la taille de la fenêtre de votre application, placez le curseur de la souris en bas à droite de la zone grise représentant le fond de votre interface. Lorsque apparaît un curseur en forme d'angle droit, vous pouvez procéder de deux façons différentes :

1. Cliquer et tirer l'angle pour augmenter ou diminuer la taille de la fenêtre.
2. Double-cliquer sur l'angle. Une boîte de dialogue apparaît affichant la largeur et la hauteur de la fenêtre de l'application. Vous pouvez modifier directement ces valeurs.

Copyright © 2016 Eyrolles.

Figure 12-18 *Ajuster la taille des composants*

Question Comment centrer le bouton Calculer dans la fenêtre d'application ?

Réponse Pour centrer le bouton Calculer :

1. Sélectionnez l'intégralité des composants de l'application puis cliquez droit sur la sélection.
2. Choisissez l'item Aligner puis Center to Column.

Définir le comportement des objets graphiques

Tous les éléments graphiques sont maintenant en place, mais l'application ne réalise pas encore de calcul. Pour cela, nous allons devoir « entrer » dans le code de l'application afin d'y insérer les instructions qui vont permettre de calculer puis d'afficher le périmètre du cercle.

Le code autogénéré par NetBeans

Dans un premier temps, examinons le code construit par NetBeans (voir figure 12-19) dans le panneau Source.

Le code, tel qu'il se présente lorsqu'on clique sur l'onglet Source, semble relativement simple. En réalité, il est plus complexe qu'il n'y paraît. Une grande partie du code est cachée. Pour le visualiser, il vous suffit de cliquer sur les signes + se trouvant en marge gauche de l'éditeur.

Sans entrer dans le détail du code généré par NetBeans, nous allons décomposer sa structure (voir figure 12-19-❶ à ❹) pour mieux comprendre son fonctionnement.

❶ Les composants graphiques créés par simple glisser-déposer de la bibliothèque vers le fond de l'application sont déclarés comme propriété de l'application, à la fin de la définition de la classe `CercleSwing`. Nous retrouvons ici les noms `calculBtn`, `perimetreLabel`, etc., que nous avons pris soin de définir dans le panneau Inspector à l'étape précédente (voir la section « Mise en place des éléments graphiques »).

Copyright © 2016 Eyrolles.

❷ La fonction `main()` est, comme nous en avons pris l'habitude tout au long de ce livre, le point d'entrée du programme. Ici, les instructions qui la composent sont cependant un peu plus complexes. Nous allons tenter de les rendre compréhensibles en les examinant plus précisément.

La fonction `main()` utilise deux outils dont nous n'avons pas encore parlé : `invokeLater()` et `Runnable`.

Ces deux outils font référence à la notion de *thread* que l'on traduit en français par « unités de traitement ».

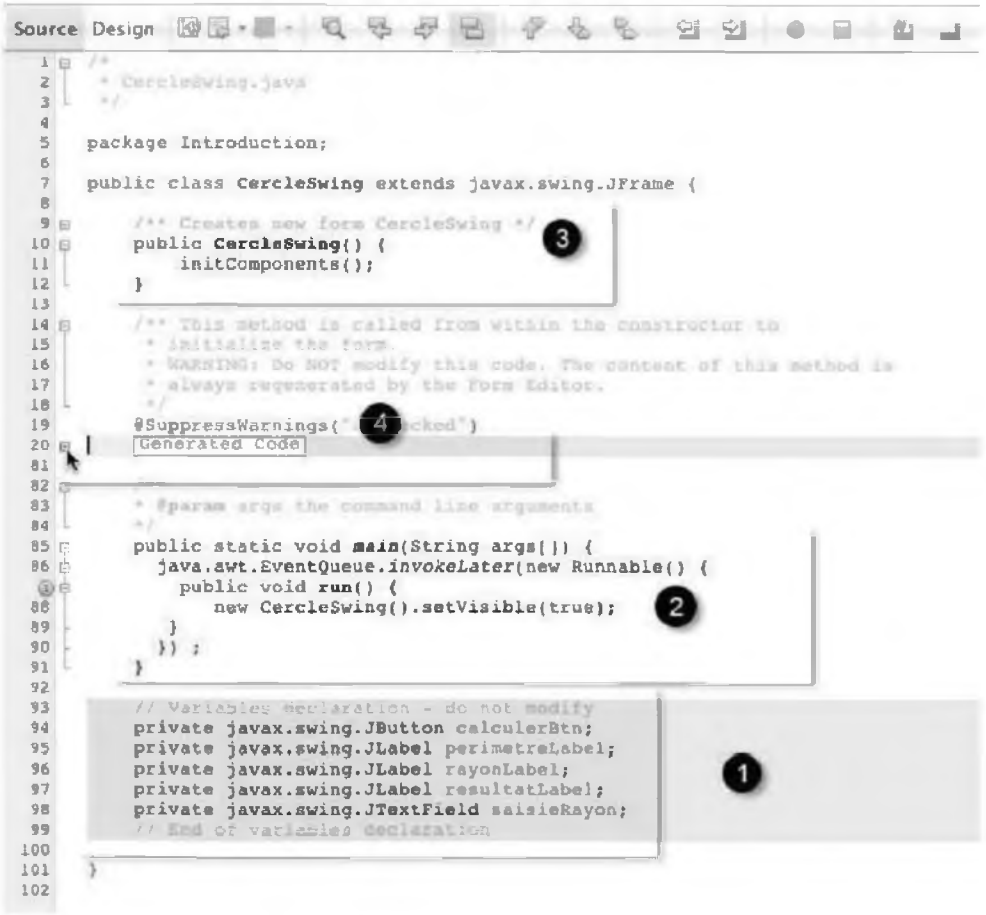


Figure 12-19 *Code source de l'application CercleSwing généré par NetBeans*

Remarque Un thread est une portion de code, un traitement spécifique capable de s'exécuter en même temps que d'autres traitements.

Copyright © 2016 Eyrolles.

Les applications construites à l'aide d'interfaces graphiques utilisent la notion de thread puisqu'elles exécutent plusieurs traitements en même temps et ce, de façon indépendante. Ces traitements sont, par exemple, la saisie d'une valeur dans un champ de saisie en même temps que l'affichage d'une aide contextuelle dans une fenêtre dédiée.

Pour exécuter les différents thread, la bibliothèque graphique Swing utilise une pile d'événements afin de répartir dans le temps leur traitement. La bibliothèque Swing propose également plusieurs méthodes pour traiter cette pile. Ici, par exemple, la méthode utilisée est `invokeLater()`, laquelle permet de traiter les événements de façon asynchrone pour éviter de bloquer les éventuels autres threads en attente.

La méthode `invokeLater()` demande, en paramètre, la portion de code à traiter de façon asynchrone. C'est ce qui est réalisé par la suite d'instructions :

```
new Runnable() {
   public void run() {
       new CercleSwing().setVisible(true);
   }
}
```

`Runnable` est une classe abstraite, une interface dont le mode de comportement défini par le programmeur doit être écrit au sein de la méthode `run()`.

Pour en savoir plus Les notions de classe abstraite et d'interface sont décrites au chapitre 9, « Dessiner des objets », section « Les événements ».

Pour notre application, le comportement à exécuter en mode asynchrone consiste à appeler le constructeur `CercleSwing()`. La classe `CercleSwing` héritant des qualités et méthodes de la classe `JFrame` (`extends javax.swing.JFrame`), l'objet issu du constructeur est une fenêtre qui devient visible grâce à l'appel de la méthode `setVisible(true)`.

Plus classiquement, nous aurions pu écrire la fonction `main()` comme suit :

```
public static void main(String args[]) {
   // Définition du thread et de son comportement
   Runnable traitement = new Runnable() {
     public void run() {
        // Création d'une fenêtre définie par la classe
        // CercleSwing
        CercleSwing fenetre = new CercleSwing();
        // Rendre visible la fenêtre
        fenetre.setVisible(true);
     };
   };
   // Traiter le thread traitement en mode asynchrone
   java.awt.EventQueue.invokeLater(traitement);
```

Copyright © 2016 Eyrolles.

```
}
```

❸ Le constructeur `CercleSwing()` appelle la méthode `initComponents()`, laquelle est entièrement générée par NetBeans.

❹ La méthode `initComponents()` n'est pas directement visible lorsqu'on clique sur l'onglet Source. Pour examiner l'intégralité du code, cliquez sur le signe + situé en marge gauche, à la hauteur du texte `Generated Code`.

Nous ne nous attarderons pas sur ce code. Il est long, fastidieux et ce seul chapitre ne suffirait pas à le décrire entièrement. Pour simplifier, il suffit juste de comprendre que l'ensemble des instructions qui le composent permet le placement et l'affichage des composants tels que vous les avez souhaités dans le panneau Design.

Associer une action à un bouton

Une fois compris le code autogénéré par NetBeans, examinons comment réaliser le calcul du périmètre d'un cercle. Celui-ci s'effectue lorsqu'on clique sur le bouton Calculer. Pour associer l'action de calcul au clic sur le bouton, nous devons ajouter un gestionnaire d'événements au bouton Calculer (nommé `calculerBtn`).

Pour en savoir plus La notion d'événements est traitée à la section « Les événements » du chapitre 9, « Dessiner des objets ».

Sous NetBeans, l'ajout d'un gestionnaire sur un composant graphique s'effectue comme suit (voir figure 12-20).

- Dans le panneau Design, sélectionnez le bouton Calculer.
- Cliquez droit sur le composant.
- Sélectionnez l'item Événements, puis Action, puis actionPerformed, dans l'enchaînement de menu et sous menus qui apparaissent.

Figure 12-20 *Associer un événement au bouton Calculer*

La fenêtre Source s'affiche et laisse apparaître une nouvelle méthode :

```
private void calculerBtnActionPerformed(java.awt.event.ActionEvent
evt) {
  // TODO add your handling code here:
}
```

Copyright © 2016 Eyrolles.

Les instructions qui réalisent le calcul du périmètre d'un cercle sont à insérer dans la méthode `calculerBtnActionPerformed()` créée par NetBeans. Ces instructions sont au nombre de trois :

1. Récupérer la valeur du rayon saisie dans le champ de saisie `saisieRayon`.
2. Calculer le périmètre.
3. Afficher le périmètre dans le champ de texte `resultatLabel`.

Ces instructions s'écrivent en Java :

```
// 1 Récupérer la valeur du rayon saisie dans le champ de saisie
//    saisieRayon
double rayon = Double.parseDouble(saisieRayon.getText());
// 2 Calculer le périmètre
double perimetre = 2* Math.PI* rayon;
// 3 Afficher le périmètre dans le champ de texte resultatLabel
resultatLabel.setText(Double.toString(perimetre));
```

Les deux principales méthodes à retenir sont : `getText()` et `setText()`.

La méthode `getText()` retourne, sous forme de chaîne de caractères, la valeur saisie par l'utilisateur dans le champ de saisie sur lequel est appliquée la méthode. Ici, il s'agit du champ `saisieRayon`. La valeur retournée par la méthode est de type `String` alors que le rayon est une valeur numérique de type `double`. L'utilisation de la méthode `parseDouble()` permet la transformation d'une chaîne de caractères en valeur de type `double`.

À l'inverse, la méthode `setText()` affiche le texte placé en paramètre dans le champ de texte sur lequel est appliqué la méthode. Ici, il s'agit du texte correspondant à la valeur calculée du périmètre. Cette valeur est numérique, la méthode `toString()` transforme cette valeur en une chaîne de caractères.

Pour en savoir plus La méthode `toString()` est également étudiée à la section « Rechercher le code Unicode d'un caractère donné » du chapitre 4, « Faire des répétitions ».

Les trois lignes de code présentées ci-avant sont à insérer dans la fonction `calculerBtnActionPerformed()` comme suit :

```
private void calculerBtnActionPerformed(java.awt.event.ActionEvent
                                        evt) {
  // Récupérer la valeur du rayon saisie dans le champ de saisie
  // saisieRayon
  double rayon = Double.parseDouble(saisieRayon.getText());
  // Calculer le périmètre
  double perimetre = 2 * Math.PI * rayon;
  // Afficher le périmètre dans le champ de texte resultatLabel
  resultatLabel.setText(Double.toString(perimetre));
}
```

Copyright © 2016 Eyrolles.

Remarque La méthode `calculerBtnActionPerformed()` est une méthode particulière que l'on nomme dans le jargon de la programmation événementielle, un gestionnaire d'événements.

Exécuter l'application

Pour voir enfin s'afficher l'application `CercleSwing`, vous devez lancer l'exécution du programme. Pour cela, cliquez sur le petit triangle vert situé au centre de la barre d'outils de NetBeans ou appuyez sur la touche F6 de votre clavier.

Si aucune erreur de compilation ou d'exécution n'est détectée, vous devez voir apparaître la fenêtre avec un rayon et un périmètre égaux à 0 (voir figure 12-21-❶). Après saisie d'une valeur pour le rayon et validation en cliquant sur le bouton Calculer, le résultat s'affiche à la suite du label « Périmètre : » (voir figure 12-21-❷).

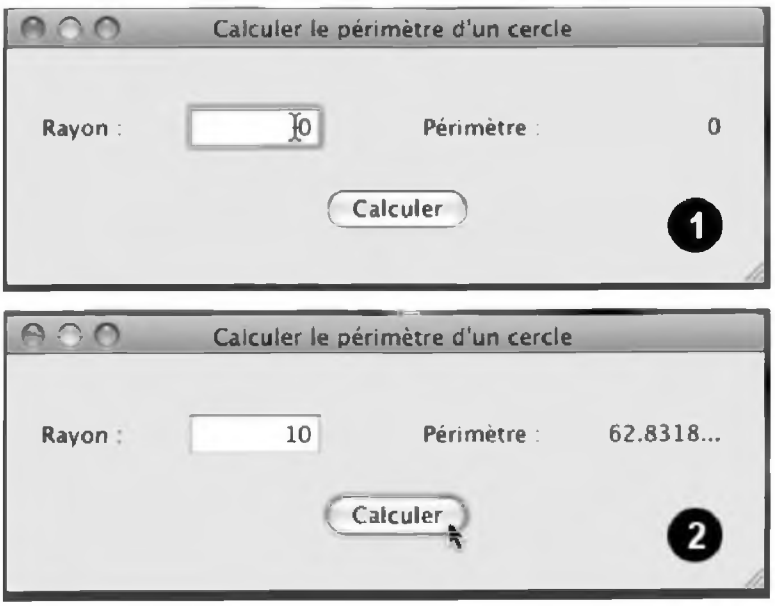


Figure 12-21 *Exécution de l'application CercleSwing.java*

Gestion de bulletins de notes

Pour parfaire nos connaissances en construction d'applications Java munies d'interfaces graphiques conviviales, nous vous proposons de créer une application permettant la création et la gestion de bulletins de notes d'étudiants.

Copyright © 2016 Eyrolles.

Remarque Pour comprendre la façon dont l'application traite les données (création d'un étudiant, sauvegarde dans un fichier d'objets, etc.), il est conseillé d'étudier le chapitre 10, « Collectionner un nombre indéterminé d'objets », avant de vous lancer dans la lecture de cette section.

Cahier des charges

L'objectif est d'écrire une application qui permet d'éditer le bulletin de notes d'un étudiant donné.

Au lancement de l'application, une première fenêtre s'affiche à l'écran (voir figure 12-22) et permet à l'utilisateur de :

1. saisir le nom et le prénom de l'étudiant dont on souhaite enregistrer les notes ;
2. choisir le semestre pour lequel les notes sont validées ;
3. rechercher sur l'ordinateur la photo de l'étudiant concerné ;
4. une zone Info située en bas de la fenêtre fournit une aide à l'utilisateur.

Ensuite, l'utilisateur peut valider la saisie ou quitter l'application.

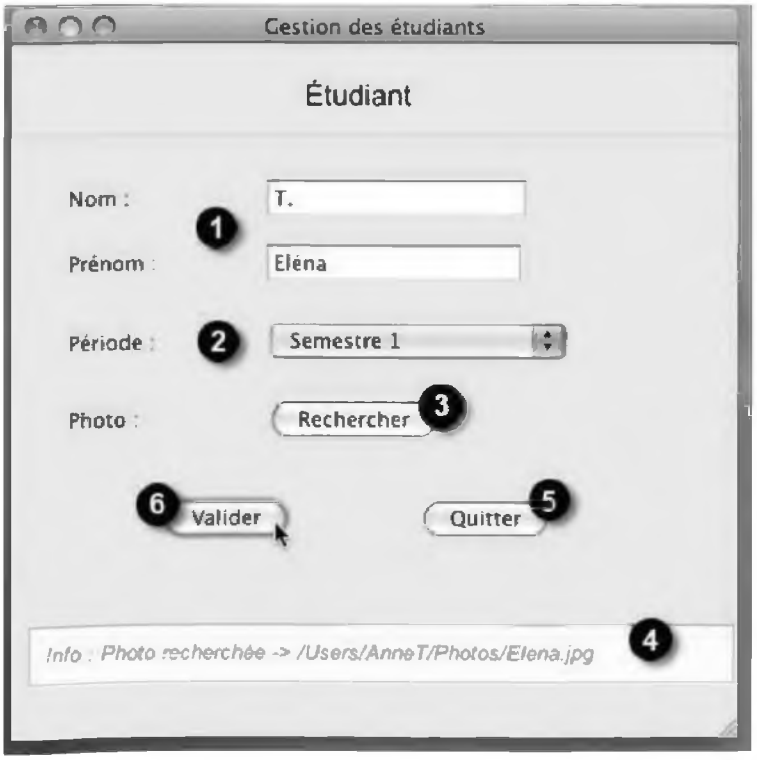


Figure 12-22 *Panneau de saisie des informations relatives à un étudiant*

Copyright © 2016 Eyrolles.

❺ si l'utilisateur clique sur le bouton Quitter (voir figure 12-22-❺), l'application cesse son exécution et la fenêtre disparaît.

❻ s'il clique sur le bouton Valider (voir figure 12-22-❻), une seconde fenêtre apparaît (voir figure 12-23). Il s'agit de la fenêtre d'édition du bulletin de notes.

Figure 12-23 *Fenêtre de saisie des notes de l'étudiant*

Le bulletin de notes affiche la photo, les nom et prénom de l'étudiant ainsi que le semestre concerné (voir figure 12-23-❶). Il affiche également les matières correspondant au semestre choisi (voir figure 12-23-❷). Pour chaque matière, un champ de saisie de la note obtenue est proposé (voir figure 12-23-❸).

La moyenne générale de l'étudiant est calculée lorsque l'utilisateur clique sur le bouton « Moyenne ? » (voir figure 12-23-❹). Le bouton Enregistrer (voir figure 12-23-❺) permet de sauvegarder l'intégralité des données dans un fichier d'objets. Le bulletin de notes disparaît lorsque l'utilisateur clique sur le bouton Fermer (voir figure 12-23-❻). Il est alors possible de créer un bulletin de notes pour un nouvel étudiant.

Copyright © 2016 Eyrolles.

Mise en place des éléments graphiques

Les fenêtres Gestion des étudiants (voir figure 12-22) et Bulletin de notes (voir figure 12-23) sont construites à partir des classes `CursusSwing` et `BulletinNote`. Elles contiennent des composants graphiques de type différents. Examinons pour chacune des classes associées, leur organisation ainsi que leur propriété.

Remarque Toutes les classes développées pour réaliser l'application `CursusSwing` sont enregistrées au sein du projet `GestionClasseExemple`, dans le package `Introduction`.

La classe CursusSwing

La classe `CursusSwing` affiche la première fenêtre demandant la saisie du nom et du prénom de l'étudiant. Elle est le point d'entrée de notre application et contient la fonction `main()`.

Structure

La fenêtre de l'application `CursusSwing.java` est construite à partir du composant `JFrame`. Ce composant contient tous les éléments graphiques (label, texte de saisie, bouton, etc.) qui permettent d'obtenir une fenêtre telle que celle présentée en figure 12-22.

La fenêtre Inspecteur (voir figure 12-24) de NetBeans résume assez bien la structure et le nombre de composants utilisés.

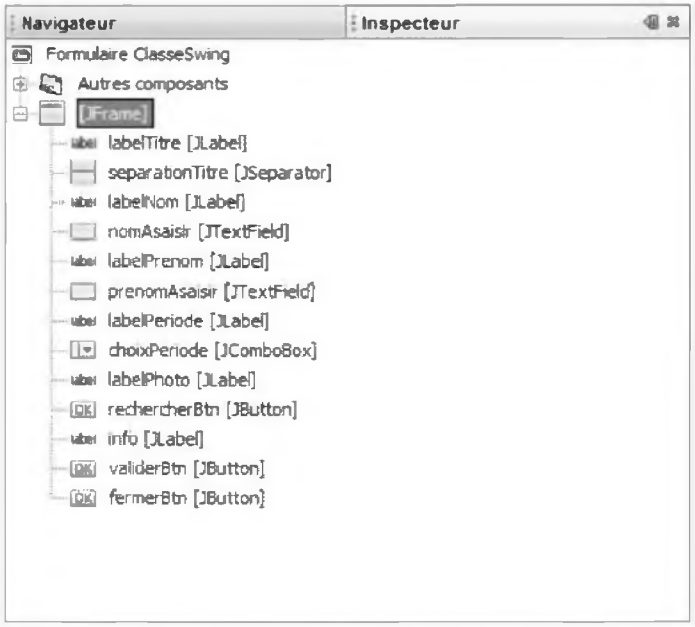


Figure 12-24 Structure de la fenêtre associée à la classe CursusSwing

Copyright © 2016 Eyrolles.

Le composant JFrame

La classe `CursusSwing` est construite à partir du composant `JFrame` dont les propriétés sont les suivantes (voir figure 12-25) :

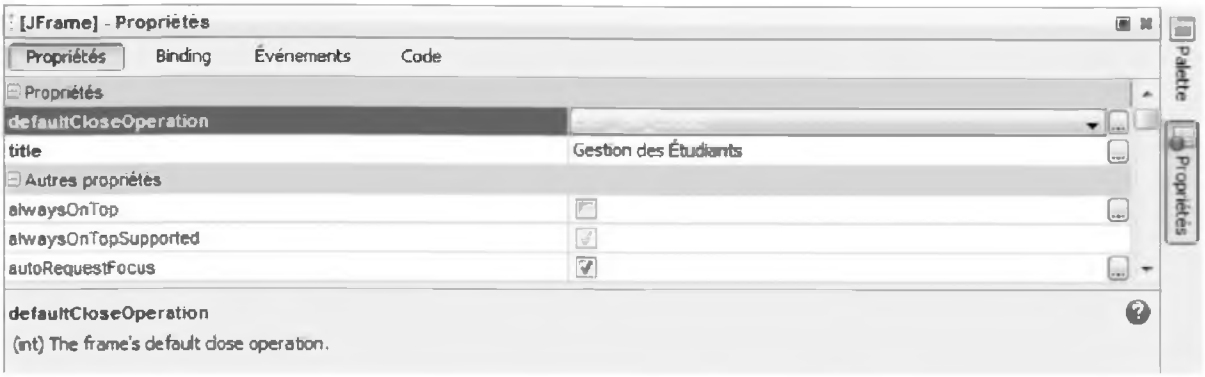


Figure 12-25 *Propriétés du composant JFrame (CursusSwing.java)*

Le titre de la fenêtre est défini à l'aide de l'attribut `title`. Dans la fenêtre Propriétés associé au composant `JFrame`, nous modifions le champ `title` pour l'initialiser à « Gestion des étudiants ».

Le champ `defaultCloseOperation` est initialisé à `EXIT_ON_CLOSE`. De cette façon, l'application finit son exécution lorsque l'utilisateur clique sur le bouton de fermeture de la fenêtre.

Les composants JLabel

Les composants `JLabel` de la fenêtre Gestion des étudiants sont au nombre de six.

Le premier `JLabel` (voir figure 12-26) est utilisé pour afficher le titre « Etudiant » dans la partie supérieure du panneau. Nous le nommons `labelTitre`, dans la fenêtre Inspecteur et insérons le texte `Etudiant` dans le champ `text` du composant.

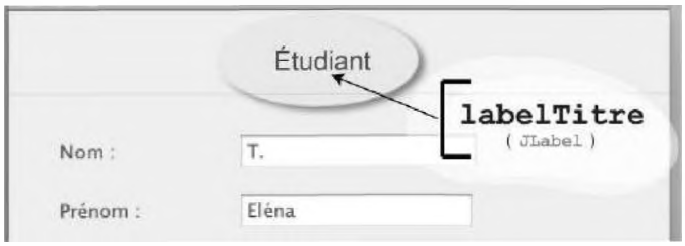


Figure 12-26 *Le composant labelTitre*

Copyright © 2016 Eyrolles.

Les quatre `JLabel` suivants (voir figure 12-27) se situent graphiquement sur les mêmes lignes que les champs de saisie auxquels ils sont logiquement associés. Par exemple, le composant `JLabel` nommé `labelNom` et dont le champ `text` est « Nom : » se situe sur la même ligne que le champ de saisie qui sera utilisé pour saisir le nom de l'étudiant. Les autres `JLabel` sont nommés respectivement `labelPrenom`, `labelPeriode` et `labelPhoto`. Ils contiennent dans l'ordre les textes « Prénom : », « Période : » et « Photo : ».

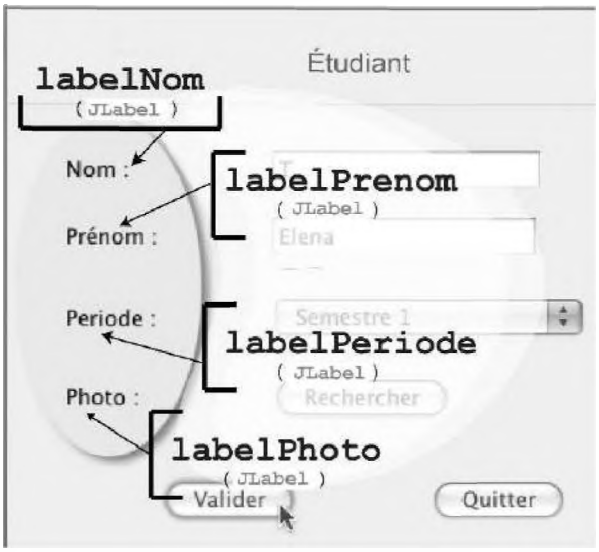


Figure 12-27 *Les composants labelNom, labelPrenom, labelPeriode et labelPhoto*

Le dernier `JLabel` (voir figure 12-28) concerne la zone Info. Nous le nommons `info` et insérons le texte « Info : » dans le champ `text` du composant.

Figure 12-28 *Le composant info*

Pour différencier l'aide contextuelle du reste de l'interface, nous modifions les propriétés `background`, `foreground` et `font` dans la fenêtre Propriétés associée au composant. Les valeurs choisies sont celles décrites en figure 12-29.

Copyright © 2016 Eyrolles.

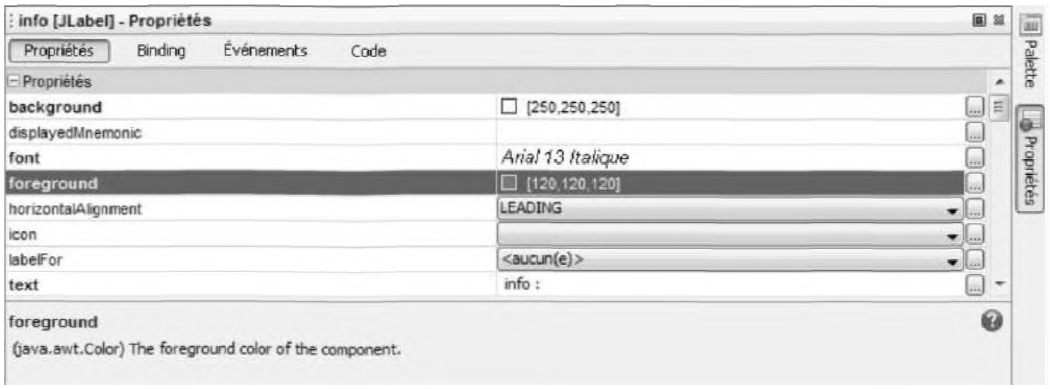


Figure 12-29 *Propriétés du composant JLabel info (BulletinNotes.java)*

Les composants JTextField

Les composants `JTextField` sont au nombre de deux. Ils permettent la saisie du nom et du prénom de l'étudiant. Nous les avons nommés par l'intermédiaire du panneau Inspecteur, `nomAsaisir` et `prenomAsaisir`.

Ils sont placés respectivement sur la même ligne que les composants nommés `labelNom` et `labelPrenom`.

Le composant JComboBox

Une `JComboBox` est une liste déroulante d'items. L'affichage par défaut ne présente qu'un seul item (voir figure 12-30-❶).

Lorsque l'utilisateur clique sur le composant (voir figure 12-30-❷), la liste se déroule et il peut alors visualiser et sélectionner l'item de son choix en cliquant dessus.

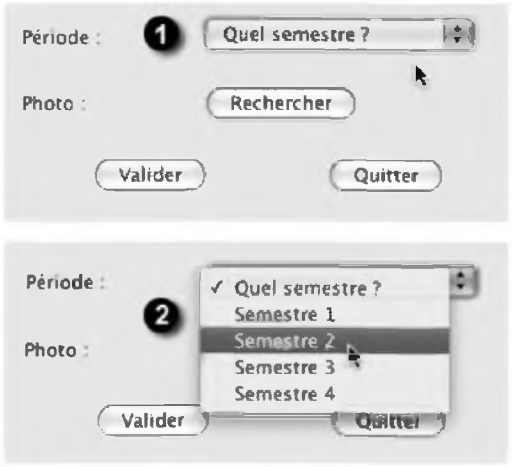


Figure 12-30 *La JComboBox*

Copyright © 2016 Eyrolles.

Pour insérer le nom des items à l'intérieur de la `JComboBox`, il vous suffit de modifier le champ `model`, dans la fenêtre Propriétés associée au composant (voir figure12-31). Ici nous avons choisi d'insérer les 5 items `Quel semestre ?`, `Semestre 1`, `Semestre 2`, `Semestre 3`, `Semestre 4`.

L'item affiché par défaut, lorsque la `JComboBox` n'est pas encore déroulée, ne correspond pas nécessairement au premier item défini dans le champ `model`.

Le champ `selectedIndex` (voir figure 12-31) permet de spécifier l'indice de l'item à afficher en premier. Cet indice varie de `0` au nombre d'items moins un, défini sur la liste construite à l'étape précédente.

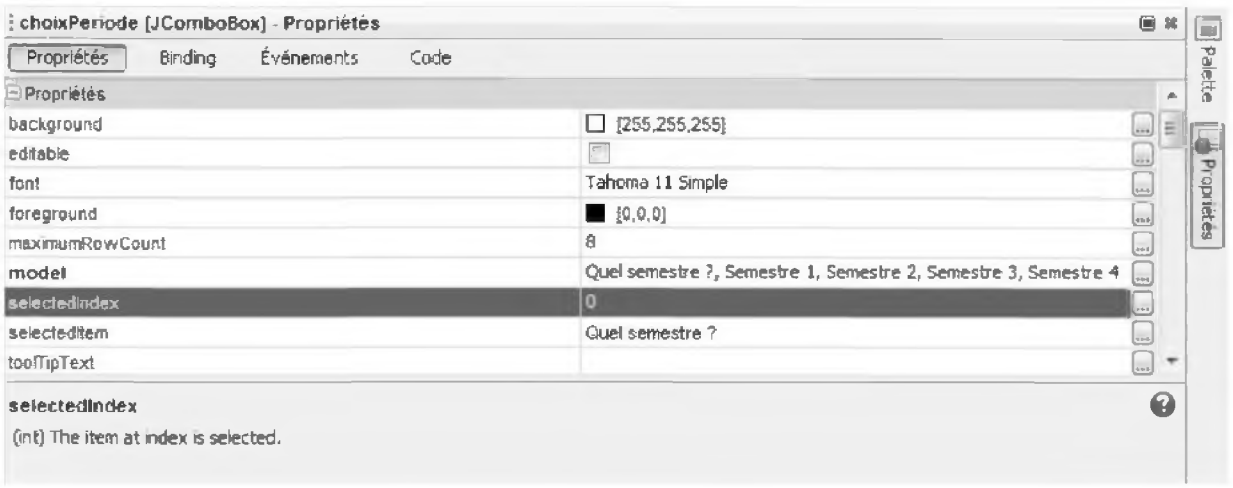


Figure 12-31 *Propriétés de la JComboBox*

Pour finir, nous nommons le composant `JComboBox`, `choixPeriode` par l'intermédiaire du panneau Inspecteur.

Les composants JButton

La fenêtre Gestion des étudiants possède deux boutons (`JButton`), nommés dans le panneau Inspecteur `validerBtn` et `fermerBtn`. Le champ `text` associé à chaque bouton est respectivement Valider et Quitter.

Les actions réalisées par les deux boutons seront décrites à la section « Définir le comportement des objets graphiques » de ce chapitre.

La classe BulletinNotes

La classe `BulletinNotes` crée la fenêtre d'édition du bulletin de notes. Elle ne s'affiche que lorsque l'utilisateur clique sur le bouton Valider de la fenêtre Gestion des étudiants.

Copyright © 2016 Eyrolles.

Structure

La fenêtre de l'application `BulletinNotes.java` est construite à partir du composant `JFrame`. Ce composant contient tous les éléments graphiques (label, texte de saisie, bouton, etc.) qui permettent d'obtenir une fenêtre telle que celle présentée en figure 12-23.

La fenêtre Inspecteur (voir figure 12-32) de NetBeans résume assez bien la structure et le nombre de composants utilisés.

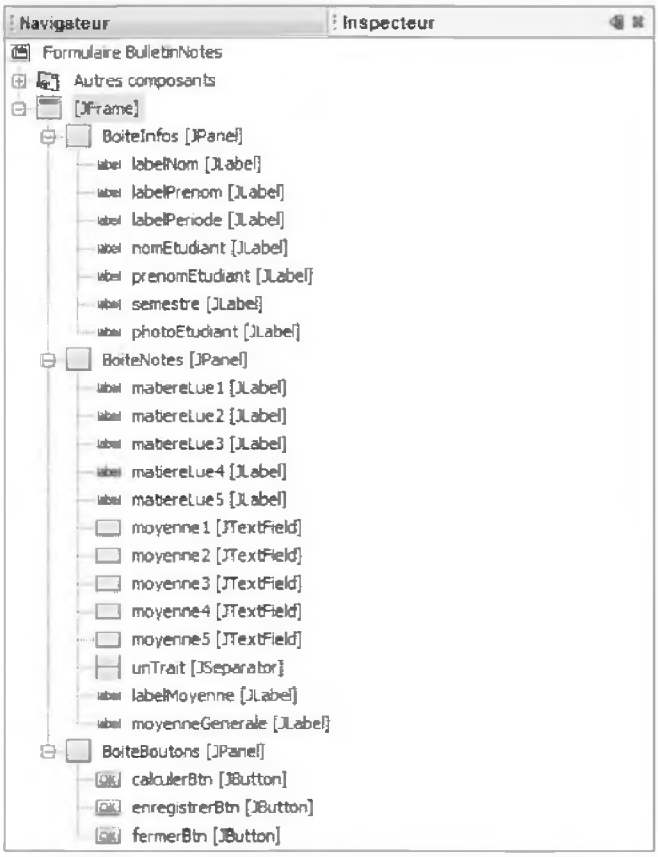


Figure 12-32 *Structure graphique de la classe BulletinNotes*

Pour simplifier la mise en place de tous ces composants, nous avons choisi de les insérer au sein de conteneurs spécifiques, les `JPanel`.

Ainsi tous les composants, situés en haut de la fenêtre, fournissent les informations relatives à l'étudiant en cours de traitement. Nous les plaçons à l'intérieur d'un composant `JPanel` nommé `BoiteInfos`.

Les composants placés au centre de la fenêtre, permettent la saisie des notes de l'étudiant. Ils sont placés dans un composant `JPanel` nommé `BoiteNotes`.

Copyright © 2016 Eyrolles.

Enfin, les boutons situés en bas de la fenêtre sont placés à l'intérieur d'un composant `JPanel` nommé `BoiteBoutons`.

Pour en savoir plus Les composants de type `JPanel` sont présentés au chapitre 11, « Dessiner des objets », section « De l'AWT à Swing ».

Le composant JFrame

La classe `BulletinNotes` est construite à partir du composant `JFrame` dont les propriétés sont les suivantes (voir figure 12-33) :

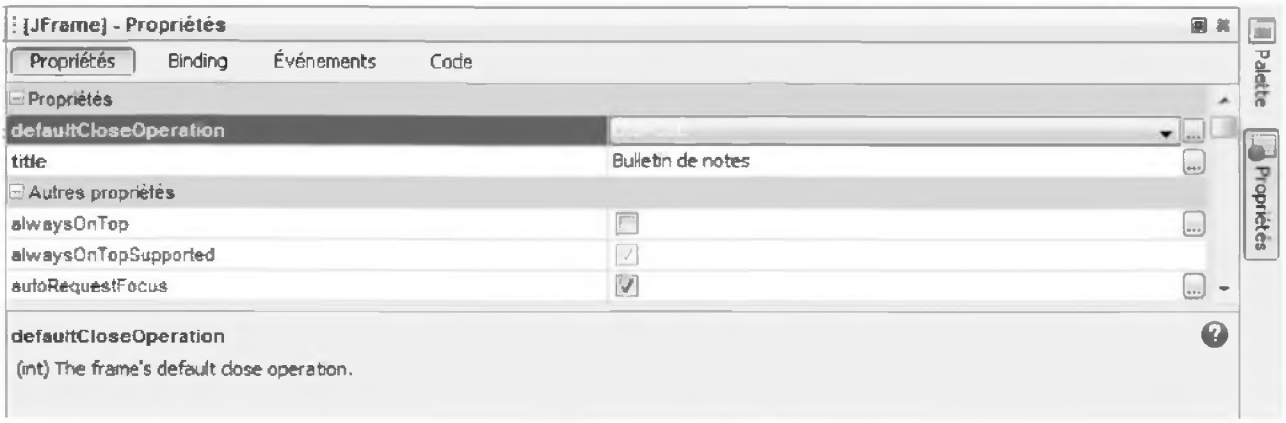


Figure 12-33 *Propriétés du composant JFrame (BulletinNotes.java)*

Le champ `title` est initialisé à « Bulletin de notes ». Le champ `defaultCloseOperation` est quant à lui initialisé à `DISPOSE`. De cette façon, l'application continue (au lieu de cesser) son exécution lorsque l'utilisateur clique sur le bouton de fermeture de la fenêtre. La fenêtre Bulletin de notes s'efface, alors que la fenêtre Gestion des étudiants reste active. L'utilisateur peut créer un nouvel étudiant.

Les composants JLabel

Les composants `JLabel` de la fenêtre Bulletin de notes sont au nombre de quatorze. Nous ne les détaillerons pas ici mais observons que certains d'entre eux restent constants alors que d'autres sont modifiés par l'application.

Les composants qui contiennent des textes qui ne seront jamais modifiés par l'application sont, par exemple, des composants contenant les textes « Nom : » ou encore « Période : ».

Copyright © 2016 Eyrolles.

Les composants modifiés par l'application sont, par exemple, les composants contenant les différentes matières associées au semestre concerné ou encore la moyenne générale obtenue à partir des notes saisies pour chaque matière.

Pour améliorer la lisibilité du code source associé, nous avons choisi la convention de nommage suivante :

- les composants restant fixes commencent par `label` suivi d'un mot qui caractérise ce à quoi il sert. Par exemple, le composant contenant « Nom : » est nommé `labelNom`, celui contenant « Période : » est appelé `labelPeriode` ;
- les composants dont le contenu est modifié par l'application portent un nom qui les caractérise. Par exemple, le composant contenant « Nom de l'étudiant » est appelé `nomEtudiant`, celui contenant « Total » est nommé `moyenneGenerale`.

La figure 12-32 présente l'ensemble des noms des composants.

Les composants JTextField

Les composants `JTextField` sont au nombre de cinq. Ils permettent la saisie des cinq notes de l'étudiant. Nous les avons nommés `moyenne1`, `moyenne2`, `moyenne3`, `moyenne4` et `moyenne5` dans le panneau Inspecteur.

Ils sont placés respectivement sur la même ligne que les composants nommés `matiereLue1`, `matiereLue2`, `matiereLue3`, `matiereLue4` et `matiereLue5`.

Les composants JButton

La fenêtre Bulletin de notes possède trois boutons, nommés `calculerBtn`, `enregistrerBtn` et `fermerBtn` dans le panneau Inspecteur. Le champ `text` associé à chaque bouton est respectivement « Moyenne ? », « Enregistrer » et « Fermer ».

Les actions réalisées par les trois boutons sont décrites à la section suivante « Définir le comportement des objets graphiques ».

Définir le comportement des objets graphiques

Les fenêtres Gestion des étudiants et Bulletin de notes s'affichent correctement. Il convient maintenant de « donner vie » aux composants afin de les voir « réagir » aux actions de l'utilisateur.

La classe CursusSwing

Plusieurs actions sont à mener pour créer un étudiant et saisir ses notes. Nous devons tout d'abord saisir toutes les informations qui le concernent (nom, prénom, photo). Ces traitements sont examinés aux sections suivantes « Mémoriser le nom et le prénom de l'étudiant » et « Rechercher la photo d'un étudiant ».

Copyright © 2016 Eyrolles.

Ensuite, la période d'enseignement pour laquel les notes sont validées doit être définie. L'opération est réalisée en sélectionnant l'item approprié dans la `JComboBox`. Son traitement sera étudié à la section suivante « Choisir la période d'enseignement ».

La zone Info est une aide contextuelle qui renseigne l'utilisateur sur ce qu'il doit faire en fonction de la position du curseur de la souris. La réalisation de l'aide contextuelle est traitée à la section suivante « La zone Info ».

Lorsque l'utilisateur a saisi tous les champs proposés par l'interface, le bulletin de notes est affiché en cliquant sur le bouton Valider. Les instructions réalisant cette action, seront examinées à la section suivante « Afficher le bulletin de notes ».

Pour finir, à la section « Quitter l'application », nous montrerons comment interrompre l'exécution de l'application, lorsque l'utilisateur clique sur le bouton Quitter.

Mémoriser le nom et le prénom de l'étudiant

Le nom et le prénom d'un étudiant sont enregistrés dans des objets de type `String`, grâce à la méthode `getText()` appliquée aux composants `nomAsaisir` et `prenomAsaisir` comme suit :

```
String nom = nomAsaisir.getText();
String prenom = prenomAsaisir.getText();
```

La mémorisation de ces valeurs n'est à effectuer que lorsque l'utilisateur clique sur le bouton Valider. Ces instructions sont donc à insérer dans la méthode `validerBtnActionPerformed()`, décrite à section suivante « Afficher le bulletin de notes » de ce chapitre.

Rechercher la photo d'un étudiant

La recherche d'une photo et la mémorisation du chemin d'accès menant à celle-ci nécessite l'utilisation d'un composant de la bibliothèque Swing nommé `JFileChooser`.

L'emploi de ce composant facilite la vie du développeur puisqu'il permet, avec très peu d'instructions, d'afficher une boîte de dialogue proposant à l'utilisateur de parcourir l'arborescence de son système de fichiers (voir figure 12-34) et de sélectionner un fichier de son choix.

Les instructions ci-après montrent comment l'utiliser :

```
// Gestionnaire d'événements associé au bouton rechercherBtn
private void rechercherBtnActionPerformed(java.awt.event.ActionEvent
                                                    evt) {
  // Création d'un objet de type File
  File fichierPhoto;
  // Création d'un composant JFileChooser
  JFileChooser fichierArechercher= new JFileChooser();
  // ❶ Ouvrir une boîte de dialogue qui permet la recherche d'un
  //    fichier
  int etatRetour = fichierArechercher.showOpenDialog(
                                          CursusSwing.this);
```

Copyright © 2016 Eyrolles.

```
    // ❷ Si l'utilisateur a sélectionné un fichier
    if (etatRetour == JFileChooser.APPROVE_OPTION) {
      // ❸ Récupérer le fichier sélectionné
      fichierPhoto = fichierArechercher.getSelectedFile();
      // ❹ Stocker le chemin d'accès au fichier dans urlPhoto
      urlPhoto = fichierPhoto.toString();
    }
}
```

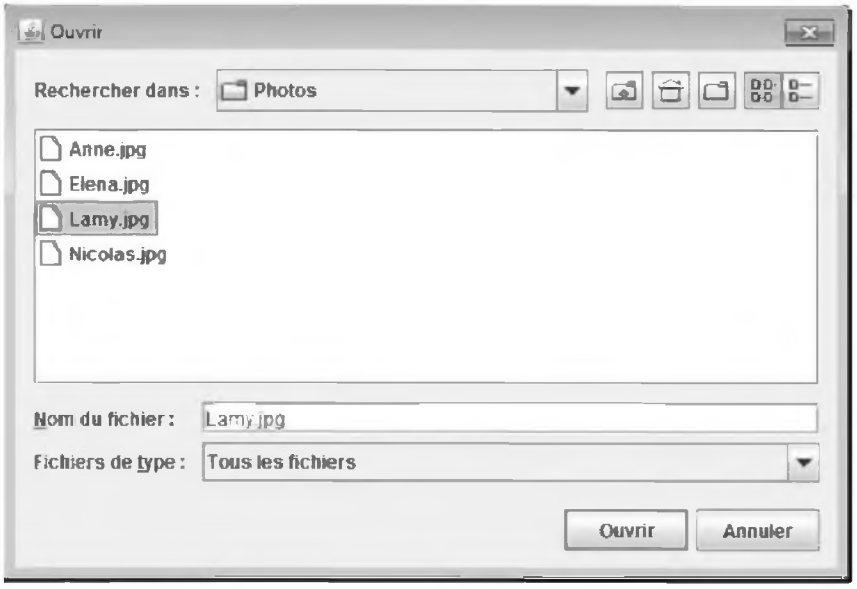


Figure 12-34 *Le composant JFileChooser*

❶ L'ouverture d'une boîte de dialogue permettant à l'utilisateur de rechercher un fichier sur son disque dur est réalisée par la méthode `showOpenDialog()`. Cette méthode doit être appliquée à l'objet `fichierArechercher` de type `JFileChooser`.

L'argument spécifié en paramètre de la méthode `showOpenDialog()` indique à la machine virtuelle Java, le nom de la fenêtre à laquelle la boîte de dialogue doit être attachée.

En utilisant le paramètre `CursusSwing.this`, nous attachons la boîte de dialogue à la fenêtre (`JFrame`) créée par l'application en cours d'exécution, c'est-à-dire la fenêtre Gestion des étudiants. De cette façon, la boîte de dialogue s'affiche obligatoirement au-dessus de la fenêtre à laquelle il est attaché.

Remarque La boîte de dialogue affiche par défaut tous les fichiers et répertoires contenus dans le répertoire personnel de l'utilisateur.

Copyright © 2016 Eyrolles.

Après avoir parcouru l'arborescence de son choix et sélectionné le fichier souhaité, l'utilisateur valide son choix en cliquant sur le bouton Ouvrir. Cela fait, la méthode `showOpenDialog()` retourne une valeur qui indique au programme que le fichier a bien été sélectionné.

❷ Si un fichier est sélectionné, la valeur retournée par la méthode `showOpenDialog()` correspond à une constante prédéfinie de la classe `JFileChooser`, `JFileChooser.APPROVE_OPTION`.

❸ Dans ce cas, le fichier sélectionné est récupéré grâce à la méthode `getSelectedFile()` appliquée à l'objet `fichierArechercher`. L'adresse du fichier est mémorisée dans un objet de type `File` et nommé `fichierPhoto`.

❹ Le chemin d'accès au fichier est enregistré dans la chaîne de caractères `urlPhoto` en appliquant la méthode `toString()` à l'objet `fichierPhoto`.

La boîte de dialogue s'ouvre seulement lorsque l'utilisateur clique sur le bouton `rechercherBtn`. L'ensemble des instructions est donc placé au sein du gestionnaire d'événements `rechercherBtnActionPerformed()`.

Pour cela vous devez :

- dans le panneau Design, sélectionner le bouton `rechercherBtn` (`JButton`) ;
- cliquer droit sur le composant ;
- sélectionner l'item Événements, puis Action, puis actionPerformed, dans l'enchaînement de menus et de sous-menus qui apparaissent ;
- la fenêtre Source s'affiche, insérer les instructions précédentes dans la méthode `rechercherBtnActionPerformed()` qui apparaît.

Le chemin d'accès au fichier photo doit être connu lorsque l'utilisateur a fini de renseigner tous les champs de la fenêtre Gestion des étudiants et surtout lorsqu'il clique sur le bouton `validerBtn`. Il convient donc de déclarer l'objet `urlPhoto` en dehors de toute fonction, comme suit :

```
private String urlPhoto="/Users/VotreNom/Photos/Inconnu.jpg";
```

La variable `urlPhoto` est définie comme variable d'instance de la classe `CursusSwing`. Elle est ainsi consultable par le gestionnaire d'événements du bouton `validerBtn`.

Pour en savoir plus Les variables d'instance sont étudiées au chapitre 7, « Les classes et les objets », section « Construire et utiliser ses propres classes ».

Nous avons volontairement initialisé la variable `urlPhoto` au chemin d'accès vers le fichier `Inconnu.jpg`. Ce fichier représentant une simple silhouette s'affiche par défaut si aucune photo n'existe pour un étudiant donné.

Copyright © 2016 Eyrolles.

Choisir la période d'enseignement

Pour sélectionner la période d'enseignement depuis la JComboBox, vous devez écrire le gestionnaire d'événements suivant :

```
// Gestionnaire d'événements associé à la JComboBox nommée choixPeriode
private void choixPeriodeActionPerformed(java.awt.event.ActionEvent
                                                    evt) {
  // ❶ Récupérer le semestre sélectionné
  periode = (String) choixPeriode.getSelectedItem();
  // ❷ Rechercher la liste des matières en fonction du semestre
  rechercherMatiere(periode);
}
```

❶ Le choix du semestre est réalisé par l'intermédiaire de l'objet choixPeriode (JComboBox), au sein du gestionnaire d'événements associé, choixPeriodeActionPerformed(). La récupération de l'item sélectionné s'effectue par l'intermédiaire de la méthode getSelectedItem() appliquée à l'objet choixPeriode.

❷ Chaque période d'enseignement possède son propre lot de matières. Au premier semestre, les matières sont, par exemple, la communication, l'anglais, l'algorithmie, la programmation et les mathématiques. Au second semestre, le multimédia remplace l'algorithmie, l'étude des réseaux se substitue aux mathématiques.

La description par matière des quatre semestres est stockée dans le fichier texte Ressources.txt dont le contenu est le suivant :

```
Semestre 1;Communication;Anglais;Algorithmie;Programmation;
Mathématiques;
Semestre 2;Communication;Anglais;Multimédia;Programmation;
Réseau;
Semestre 3;Communication;Marketing;Conduite de projet;Vidéo;
Multimédia;
Semestre 4;Communication;Management;Multimédia;Stage;Projet;
```

Chaque ligne définit le contenu d'un semestre et les intitulés de matière sont séparés par des points-virgules (;). Le premier champ indique à quel semestre correspond la ligne.

Pour lire ce type de fichier et en extraire le contenu, nous utilisons la méthode décrite au chapitre 10, « Collectionner un nombre indéterminé d'objets », section « Exercice ». La méthode rechercherMatiere() reprend le code donné en correction de l'exercice 10.5. Les instructions qui la composent sont les suivantes :

```
private void rechercherMatiere(String quelSemestre) {
  Fichier fichierTxt = new Fichier();
  listeMatieres = new String [5];
  String [] tousLesmots = new String[6];
  if (fichierTxt.ouvrir("/Users/VotreNom/Ressources/
                        Ressources.txt", "R")) {
```

Copyright © 2016 Eyrolles.

```
        do {
          tousLesmots = fichierTxt.lire();
          if (tousLesmots[0].equalsIgnoreCase(quelSemestre)) {
             for (int i=1; i < tousLesmots.length; i++) {
                listeMatieres[i-1] = tousLesmots[i];
             }
             break;
          }
        } while (tousLesmots[0]!= null);
      fichierTxt.fermer();
      }
}
```

Extension Web Vous trouverez les explications, ligne par ligne, du code dans le fichier `Corriges.pdf` sur l'extension Web de l'ouvrage.

La méthode `rechercherMatiere()` lit le fichier `Ressources.txt` pour en extraire la liste des matières, en fonction du semestre passé en paramètre. Cette liste est stockée dans un tableau nommé `listeMatieres`, lequel est déclaré en dehors de toute fonction, comme suit :

```
private String [] listeMatieres;
```

Le tableau `listeMatieres` est défini comme variable d'instance de la classe `CursusSwing`. Il est consultable par le gestionnaire d'événements associé au bouton Valider.

La zone Info

La zone Info est une aide contextuelle qui affiche un message différent selon la position du curseur de la souris. Par défaut, le message affiché dans la zone Info, est « Info : Tous les champs doivent être renseignés ! ».

Si le curseur de la souris se trouve :

- sur le champ de saisie du nom de l'étudiant, le message affiché dans la zone Info est « Info : Saisir le nom de l'étudiant » ;
- sur le champ de saisie du prénom de l'étudiant, le message affiché dans la zone Info est « Info : Saisir le prénom de l'étudiant » ;
- sur le bouton Rechercher, le message affiché dans la zone Info est « Info : Rechercher la photo de l'étudiant » ;
- sur le bouton Valider, le message affiché dans la zone Info est « Info : Afficher le bulletin de notes de l'étudiant » ;
- sur le bouton Quitter, le message affiché dans la zone Info est « Info : Quitter l'application ».

Le texte s'affiche dans la zone Info lorsque le curseur de la souris survole un composant et non suite à un clic de souris. La mise en place d'un gestionnaire d'événements lié au survol de la souris s'effectue de la façon suivante (voir figure 12-35).

Copyright © 2016 Eyrolles.

- Dans le panneau Design, sélectionnez, par exemple, le composant nommé `nomAsaisir` (`JTextField`).
- Cliquez droit sur le composant.
- Sélectionnez l'item Événements, puis Mouse, puis mouseEntered, dans l'enchaînement de menus et de sous-menus qui apparaissent.

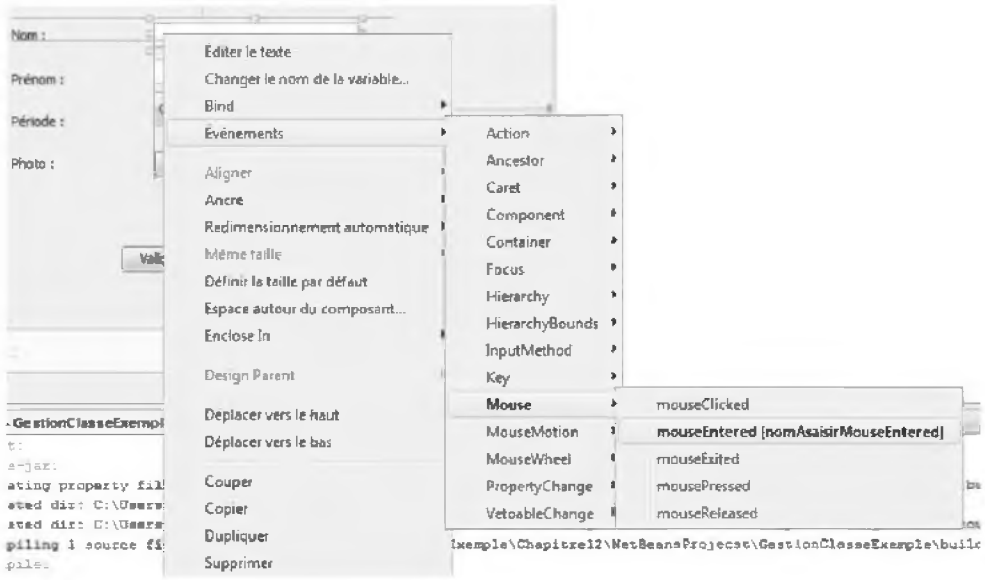


Figure 12-35 *Création d'un gestionnaire de type mouseEntered*

Le panneau Source s'affiche et laisse apparaître une nouvelle méthode :

```
private void nomAsaisirMouseEntered(java.awt.event.ActionEvent evt) {
// Insérer ici le code qui affiche un texte dans la zone Info
}
```

Cette opération est à réaliser pour tous les composants concernés par le survol de la souris. Nous obtenons au final, le code source des gestionnaires d'événements suivant :

```
// Survol du champ de saisie nomAsaisir
private void nomAsaisirMouseEntered(java.awt.event.MouseEvent evt) {
  Info.setText("Info : Saisir le nom de l'étudiant");
}

// Survol du champ de saisie prenomAsaisir
private void prenomAsaisirMouseEntered(java.awt.event.MouseEvent
                                       evt) {
  Info.setText(" Info : Saisir le prénom de l'étudiant");
}
```

Copyright © 2016 Eyrolles.

```
// Survol du bouton rechercherBtn
private void rechercherBtnMouseEntered(java.awt.event.MouseEvent
                                        evt) {
  Info.setText(" Info : Rechercher la photo de l'étudiant");
}
// Survol du bouton validerBtn
private void validerBtnMouseEntered(java.awt.event.MouseEvent evt) {
  Info.setText("Info : Afficher le bulletin de notes de L'étudiant");
}
// Survol du bouton quitterBtn
private void fermerBtnMouseEntered(java.awt.event.MouseEvent evt) {
  Info.setText("Info : Quitter l'application");
}
```

Pour faire en sorte que la zone Info affiche par défaut le message « Info : Tous les champs doivent être renseignés ! », il suffit de placer un gestionnaire de type `mouseEntered` sur le fond de la fenêtre, soit le composant `JFrame`.

Le code source du gestionnaire d'événements s'écrit alors comme suit :

```
private void formMouseEntered(java.awt.event.MouseEvent evt) {
  Info.setText("Info : Tous les champs doivent être renseignés !");
}
```

Afficher le bulletin de notes

Lorsque l'utilisateur a renseigné tous les champs de la fenêtre Gestion des étudiants, il peut enfin cliquer sur le bouton Valider afin de visualiser le bulletin de notes de l'élève concerné.

L'affichage du bulletin de notes est donc réalisé par le gestionnaire `validerBtnActionPerformed()`. Les instructions qui le composent sont les suivantes :

```
private void validerBtnActionPerformed(java.awt.event.ActionEvent
                                        evt) {
  // ❶ Enregistrer les nom et prénom de l'étudiant
  String nom = nomAsaisir.getText();
  String prenom = prenomAsaisir.getText();
  // Créer et initialiser à 0 le tableau des moyennes
  double [] moyenne = new double[listeMatieres.length];
  for (int i=0; i < moyenne.length; i++) moyenne[i]=0;
  // ❷ Créer un étudiant
  Etudiant eleve = new Etudiant(nom, prenom, urlPhoto,
                                listeMatieres, periode, moyenne);
  // ❸ Créer un bulletin de notes
  BulletinNotes bn = new BulletinNotes(eleve);
}
```

Copyright © 2016 Eyrolles.

Toutes les données nécessaires à la construction du bulletin de notes sont maintenant en place.

❶ Le nom et le prénom sont initialisés au nom et au prénom saisis par l'intermédiaire des champs de saisie. Un tableau des moyennes est créé puis initialisé à 0.

❷ Les données relatives à l'étudiant en cours de traitement sont rassemblées dans un seul objet nommé `eleve`. Elles sont enregistrées par l'intermédiaire du constructeur de la classe `Etudiant`.

Extension Web La classe `Etudiant` utilisée ici, reprend en grande partie le code de la classe `Etudiant` décrite en section « Trier un ensemble de données » du chapitre 9, « Collectionner un nombre fixe d'objets ». Pour plus de précisions sur les modifications, reportez-vous au fichier `Etudiant.java` se trouvant dans le répertoire `Source/Exemples/Chapitre12/NetBeans/BulletinDeNotes` sur l'extension Web de l'ouvrage.

❸ Le bulletin de notes est ensuite affiché grâce au constructeur de la classe `BulletinNotes`. Toutes les informations nécessaires à l'affichage des données de l'étudiant sont passées en paramètres du constructeur, par l'intermédiaire de l'objet `eleve`.

Nous étudions la classe `BulletinNotes` à la section « La classe BulletinNotes » ci-après.

Quitter l'application

Pour quitter l'application, l'utilisateur clique sur le bouton Quitter. Le gestionnaire d'événements associé à ce bouton s'écrit tout simplement comme suit :

```
private void quitterBtnActionPerformed(java.awt.event.ActionEvent
                                                    evt) {
  System.exit(0);
}
```

La classe BulletinNotes

La création d'un bulletin de notes a pour résultat d'afficher une fenêtre contenant toutes les informations relatives à un étudiant donné. Ces informations sont transmises de l'application `ClassSwing` au bulletin de notes par l'intermédiaire du constructeur `BulletinNotes()`. Les instructions qui le composent seront étudiées à la section suivante « Afficher le nom, le prénom et la photo de l'étudiant ».

L'étude de la section « Saisir les notes et calculer la moyenne » vous montrera comment récupérer les notes et calculer la moyenne lorsque l'utilisateur clique sur le bouton Moyenne ?

Le bouton Enregistrer permet de sauvegarder les données dans un fichier d'objets. La réalisation de cette sauvegarde sera traitée à la section « Enregistrer les informations ».

Pour finir, à la section « Fermer la fenêtre », nous expliquerons comment fermer une fenêtre sans interrompre l'exécution de l'application.

Copyright © 2016 Eyrolles.

Afficher le nom, le prénom et la photo de l'étudiant

Les données saisies dans la fenêtre Gestion des étudiants sont transmises au bulletin de notes par l'intermédiaire de l'objet de type `Etudiant` passé en paramètre du constructeur.

Le constructeur `BulletinNotes()` est composé des instructions suivantes :

```
public BulletinNotes(Etudiant tmp) {
    // Rendre le panneau visible
    this.setVisible(true);
    // Mettre en place les composants créés dans le panneau Design
    initComponents();
    // Définir la position et la taille du panneau, à l'écran
    this.setBounds(100, 100,522, 669);
    // ❶ Initialiser la propriété etudiant a l'étudiant passé
    //    en paramètre
    etudiant = tmp;
    // ❷ Afficher les nom et prénom de l'étudiant
    nomEtudiant.setText(etudiant.getNom());
    prenomEtudiant.setText(etudiant.getPrenom());
    // Afficher le semestre d'enseignement
    semestre.setText(etudiant.getSemestre());
    // Récupérer la liste des matières
    String [] listeMatieres = etudiant.getMatieres();
    // ❸ Afficher chaque matière dans le composant qui lui
    //    correspond
    matiereLue1.setText(listeMatieres[0]);
    matiereLue2.setText(listeMatieres[1]);
    matiereLue3.setText(listeMatieres[2]);
    matiereLue4.setText(listeMatieres[3]);
    matiereLue5.setText(listeMatieres[4]);
    // ❹ Récupérer la moyenne de l'étudiant pour chaque matière
    //    La première fois, la moyenne est forcément nulle
    double [] listeMoyenne = etudiant.getMoyenne();
    // Afficher chaque moyenne dans le composant qui lui correspond
    moyenne1.setText(Double.toString(listeMoyenne[0]));
    moyenne2.setText(Double.toString(listeMoyenne[1]));
    moyenne3.setText(Double.toString(listeMoyenne[2]));
    moyenne4.setText(Double.toString(listeMoyenne[3]));
    moyenne5.setText(Double.toString(listeMoyenne[4]));
    // ❺ Afficher la photo de l'étudiant
    ImageIcon iconPhoto = new ImageIcon(etudiant.getPhoto());
    photoEtudiant.setIcon(iconPhoto);
}
```

Copyright © 2016 Eyrolles.

Les trois premières instructions du constructeur ont pour rôle d'afficher le bulletin de notes avec les composants que nous avons créés dans le panneau Design de l'interface NetBeans.

❶ Les données concernant l'étudiant dont on affiche le bulletin de notes sont transmises au constructeur par l'intermédiaire du paramètre `tmp`. Nous stockons ces données dans un objet nommé `etudiant`, déclaré comme variable d'instance de la classe `Bulletin Notes`, comme suit :

```
private Etudiant etudiant;
```

Nous pourrons ainsi consulter ou modifier ses données à tout moment.

❷ L'affichage des nom et prénom de l'étudiant est réalisé par la méthode `setText()` appliquée aux labels `nomEtudiant` et `prenomEtudiant`. La récupération du nom et du prénom est quant à elle effectuée en appelant les méthodes d'accès en consultation `getNom()` et `getPrenom()` définies dans la classe `Etudiant`.

Extension Web Les méthodes d'accès (`get` ou `set`) sont étudiées au chapitre 8, « Les principes du concept objet », section « Les méthodes d'accès aux données ». La classe Etudiant se trouve dans le répertoire `Source/Exemples/Chapitre12/NetBeans/Bulletin DeNotes` sur l'extension Web de cet ouvrage.

❸ De la même façon, les différentes matières sont affichées dans les labels `matiereLue1` à `matiereLue5`. L'intitulé des matières est extrait de l'objet `etudiant` grâce à la méthode d'accès en consultation `getMatieres()`.

❹ La moyenne d'un étudiant, par matière, vaut initialement `0`. Cette valeur sera ensuite modifiée lorsque l'utilisateur saisira de nouvelles valeurs dans les champs de saisie appropriés.

Remarque Même si récupérer des moyennes nulles semble être inutile à cette étape de la conception, cette information vous sera utile pour réaliser l'exercice 12-4 dont l'objectif est de modifier le bulletin de notes d'un étudiant déjà enregistré.

❺ Pour afficher une photo, nous utilisons le label `photoEtudiant` créé à l'aide du panneau Design. L'insertion d'une image (fichier au format JPEG) dans un composant `JLabel` est réalisée en créant un objet de type `ImageIcon` grâce à l'instruction suivante :

```
ImageIcon iconPhoto = new ImageIcon(etudiant.getPhoto());
```

Le constructeur `ImageIcon()` demande en paramètre le chemin d'accès au fichier image. Ce chemin est fourni par la méthode d'accès en consultation `getPhoto()`.

Une fois l'icône `iconPhoto` créée, elle est placée dans le label `photoEtudiant` grâce à la méthode `setIcon()`.

Copyright © 2016 Eyrolles.

Saisir les notes et calculer la moyenne

L'utilisateur transmet les moyennes par matière grâce aux champs de saisie `moyenne1` à `moyenne5`. La moyenne est calculée lorsqu'il clique sur le bouton `calculerBtn`.

L'affichage des différentes moyennes est donc réalisé par le gestionnaire `calculerBtn ActionPerformed()`. Les instructions qui le composent sont les suivantes :

```
private void calculerBtnActionPerformed(java.awt.event.ActionEvent
                                        evt) {
  // Récupérer la première moyenne
  String moyenneTxt = moyenne1.getText();
  // Transformer la chaîne de caractères en valeur numérique
  float m1 = Float.parseFloat(moyenneTxt);
  // Mémoriser la nouvelle moyenne de l'étudiant en cours
  // de traitement
  etudiant.setMoyenne(m1, 0);
  // Récupérer la deuxième moyenne
  moyenneTxt = moyenne2.getText();
  // Transformer la chaîne de caractères en valeur numérique
  float m2 = Float.parseFloat(moyenneTxt);
  // Mémoriser la nouvelle moyenne de l'étudiant en cours
  // de traitement
  etudiant.setMoyenne(m2,1);
  // Récupérer la troisième moyenne
  moyenneTxt = moyenne3.getText();
  // Transformer la chaîne de caractères en valeur numérique
  float m3 = Float.parseFloat(moyenneTxt);
  // Mémoriser la nouvelle moyenne de l'étudiant en cours
  // de traitement
  etudiant.setMoyenne(m3, 2);
  // Récupérer la quatrième moyenne
  moyenneTxt = moyenne4.getText();
  // Transformer la chaîne de caractères en valeur numérique
  float m4 = Float.parseFloat(moyenneTxt);
  // Mémoriser la nouvelle moyenne de l'étudiant en cours
  // de traitement
  etudiant.setMoyenne(m4, 3);
  // Récupérer la cinquième moyenne
  moyenneTxt = moyenne5.getText();
  // Transformer la chaîne de caractères en valeur numérique
  float m5 = Float.parseFloat(moyenne5.getText());
  // Mémoriser la nouvelle moyenne de l'étudiant en cours
  // de traitement
```

Copyright © 2016 Eyrolles.

```
    etudiant.setMoyenne(m5, 4);
    // Calculer la moyenne générale
    float resultat = (m1 + m2 + m3 + m4 + m5)/5;
    // Afficher le résultat dans le composant moyenneGenerale
    moyenneGenerale.setText(Float.toString(resultat));
}
```

Les instructions de récupération et d'affichage de données vous sont maintenant familières, nous ne les commenterons pas plus avant.

Remarquons simplement que pour calculer une moyenne, nous devons travailler avec des données numériques et non textuelles. Pour cela, il convient de transformer les valeurs saisies dans les champs de saisie (`JTextField`) en valeur numérique. La méthode `Float.parseFloat()` réalise cette opération.

À l'inverse la méthode `Float.toString()` transforme une valeur numérique en chaîne de caractères. Nous l'utilisons pour afficher la moyenne générale dans le label `moyenneGenerale`.

Enregistrer les informations

Pour enregistrer les données, l'utilisateur clique sur le bouton Enregistrer. Le gestionnaire d'événements associé à ce bouton s'écrit comme suit :

```
private void enregistrerBtnActionPerformed(java.awt.event.ActionEvent
                                           evt) {
    // Créer une classe d'étudiants
    Cursus promo = new Cursus();
    // Créer un fichier d'étudiants
    FichierEtudiant F = new FichierEtudiant();
    // Si le fichier s'ouvre en lecture
    if (F.ouvrir("L")) {
        // Lire les données et les mémoriser dans l'objet
        promo = F.lire();
        // Fermer le fichier
        F.fermer();
    }
    // Ajouter l'étudiant en cours de traitement à la promo
    promo.ajouteUnEtudiant(etudiant.getNom(), etudiant.getPrenom(),
                           etudiant.getPhoto(), etudiant.getMatieres(),
                           etudiant.getSemestre(), etudiant.moyenne);
    // Ouvrir le fichier en écriture
    F.ouvrir("W");
    // Enregistrer la promo avec l'étudiant supplémentaire
```

Copyright © 2016 Eyrolles.

```
    F.ecrire(promo);
    // Fermer le fichier
    F.fermer();
}
```

Le code d'enregistrement des données dans un fichier d'objets reprend les classes `Cursus` et `FichierEtudiant` étudiées au chapitre 10, « Collectionner un nombre indéterminé d'objets ». Pour plus de précisions sur sa mise en place et son fonctionnement, reportez-vous à la section « Les fichiers d'objets – Exemple : archiver une classe d'étudiants ».

Remarque

Pour que l'application `CursusSwing` soit compilable et s'exécute correctement, il est nécessaire d'insérer les classes `Cursus` et `FichierEtudiant` au sein du projet `GestionClasseExemple`.

Fermer la fenêtre

Pour fermer le bulletin de notes, l'utilisateur clique sur le bouton Fermer. Le gestionnaire d'événements associé à ce bouton s'écrit comme suit :

```
private void fermerBtnActionPerformed(java.awt.event.ActionEvent
                                      evt) {
    this.dispose();
}
```

La méthode `dispose()` ferme la fenêtre sur laquelle la méthode est appliquée. Elle libère toutes les ressources mémoire associées sans pour autant cesser l'exécution de l'application.

Ici, la méthode est appliquée à `this`, c'est-à-dire à la fenêtre en cours de traitement, soit Bulletin de notes.

Un éditeur pour dessiner

Cette section aborde, à travers un exemple classique, les différentes techniques d'affichage d'éléments graphiques à l'aide de la bibliothèque Swing et de NetBeans.

Remarque

Pour comprendre la façon dont sont traitées les données (création de formes graphiques, création d'une liste d'affichage, sauvegarde dans un fichier texte), il est conseillé d'étudier le chapitre 10, « Collectionner un nombre indéterminé d'objets », ainsi que le chapitre 11, « Dessiner des objets ».

L'objectif est ici d'écrire un éditeur graphique qui offre à l'utilisateur la possibilité de dessiner des cercles ou des rectangles de la couleur de son choix.

Copyright © 2016 Eyrolles.

Cahier des charges

Au lancement de l'application, la fenêtre Éditeur graphique s'affiche à l'écran sous la forme suivante (voir figure 12-36).

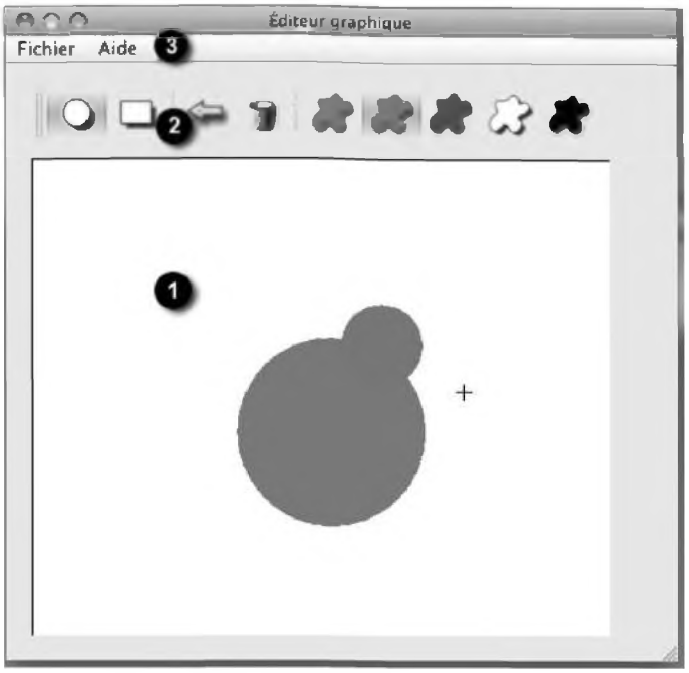


Figure 12-36 *La fenêtre Éditeur graphique*

La fenêtre de l'éditeur se divise en trois zones :

- La zone de dessin (voir figure 12-36-❶) est la feuille blanche du dessinateur. Ce dernier dessine des formes en cliquant puis en déplaçant le curseur de la souris. La forme se dessine sur la page lorsque l'utilisateur relâche le bouton de la souris. Les algorithmes permettant le dessin d'une forme à l'aide de la souris ainsi que les méthodes d'affichage de la bibliothèque Swing, seront étudiés à la section « Créer une feuille de dessins ».
- La boîte à outils (voir figure 12-36-❷) est constituée de plusieurs icônes, chacune proposant une fonctionnalité particulière comme dessiner un cercle ou un rectangle, choisir une couleur de dessin ou encore effacer tout ou la dernière forme réalisée. La mise en place de la boîte à outils et ses fonctionnalités seront traitées à la section « Créer une boîte à outils ».
- Le menu (voir figure 12-36-❸) est composé de deux items : Fichier et Aide. Le menu Fichier est utile pour ouvrir, enregistrer ou créer un dessin. Il permet aussi de quitter l'application. Le menu Aide affiche la fenêtre intitulée À propos. La mise en œuvre du menu sera expliquée à la section « Créer un menu ».

Copyright © 2016 Eyrolles.

Structure de l'application

L'application est créée sous la forme d'un projet NetBeans nommé `EditeurExemple`. Ce dernier est constitué des classes suivantes :

Pour en savoir plus La mise en place d'un projet sous NetBeans est expliquée à la section « Les bases de NetBeans – Développer une interface graphique en mode projet » au début de ce chapitre.

- La classe `Main`, qui contient, comme son nom l'indique, la fonction `main()`. Elle est construite à partir du composant `JFrame` et comprend tous les éléments graphiques (menu, barre d'outils, feuille de dessins, etc.) qui permettent d'obtenir une fenêtre telle que celle présentée en figure 12-36.
- La classe `FeuilleDeDessins`, qui est une description de la zone de dessin. Elle regroupe en son sein tous les comportements liés à l'affichage des objets, comme récupérer les coordonnées de la souris au moment du clic ou encore dessiner un cercle.
- Les classes `Forme`, `Cercle`, `Rectangle`, `ListeDeFormes` et `Fichier`. Ces classes sont celles étudiées et construites tout au long de ce livre. Nous utilisons leur version finale, écrite au chapitre 10, « Collectionner un nombre indéterminé d'objets », section « Exercices ».

Le panneau Inspecteur (voir figure 12-37) de la classe `Main` permet d'avoir une vue d'ensemble de la structure ainsi que du nombre de composants utilisés. Son examen vous fournira le nom et le type de chacun d'entre eux.

La structure générale de la fenêtre associée à la classe `Main` se divise en trois composants : `barreMenu` (`JMenubar`), `boiteDessin` (`JPanel`) et `boiteOutils` (`JToolBar`). La mise en place de ces composants est étudiée ci-après.

Créer une feuille de dessins

La feuille de dessins est décrite par une classe à part entière, la classe `FeuilleDeDessins`. C'est par son intermédiaire que les coordonnées de la souris sont récupérées ou que l'objet à dessiner est affiché sur la zone créée à cet effet, et non sur la boîte à outils par exemple.

Mise en place des éléments graphiques

Pour créer l'éditeur graphique, il convient de savoir dessiner et afficher des formes géométriques telles qu'un cercle ou un rectangle mais plus encore, nous devons définir une « structure des données » associées aux différentes formes à dessiner sur la feuille.

Sans cette structure, nous ne pourrons pas enregistrer le dessin et le voir réapparaître en ouvrant le fichier d'enregistrement. Il nous sera également impossible d'effacer la dernière forme affichée (action « undo »).

Copyright © 2016 Eyrolles.

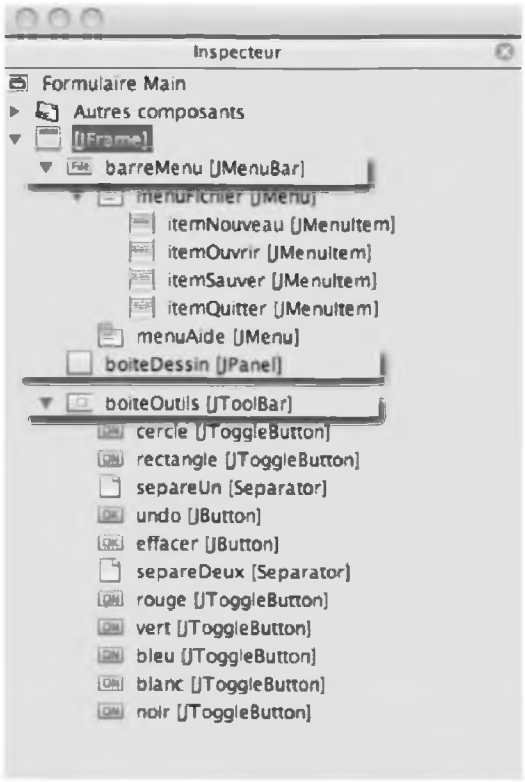


Figure 12-37 *Structure de la fenêtre associée à la classe Main*

La structure de la liste d'affichage

La structure des données reprend celle décrite à l'exercice 10-2 du chapitre 10, « Collectionner un nombre indéterminé d'objets ». Il s'agit d'enregistrer chaque forme dessinée au sein d'une liste de type `ArrayList`.

La classe `ListeDeFormes` utilise cette structure. Elle contient toutes les méthodes dont nous avons besoin, à savoir :

- Ajouter une forme à la liste. La fonction qui réalise l'ajout d'un cercle ou d'un rectangle à la liste des formes s'écrit comme suit :

```
public void ajouterUneForme(Forme f) {
   // Si la forme passée en paramètre est un Cercle,
   if (f instanceof Cercle) {
     // l'ajouter à la liste comme forme de type Cercle
     listeFormes.add((Cercle) f);
   }
   // Si la forme passée en paramètre est un Rectangle,
   else if (f instanceof Rectangle) {
```

Copyright © 2016 Eyrolles.

```
            // l'ajouter à la liste comme forme de type Rectangle
            listeFormes.add((Rectangle) f);
        }
    }
```

Remarque L'objet `listeFormes` est déclaré comme propriété de la classe `ListeDeFormes` comme étant de type `ArrayList <Forme>`.

- Supprimer toutes les formes. La suppression de toutes les formes de la liste s'effectue très simplement comme suit :

```
public void supprimerLesFormes(){
    // La méthode clear() vide la liste de son contenu
    listeFormes.clear();
}
```

- Afficher le contenu de la liste. La méthode qui consiste à afficher le contenu de la liste sous forme de texte n'est pas nécessairement utile lorsque l'on souhaite dessiner des formes graphiquement. Cependant, elle a l'avantage d'aider le développeur lors de la construction de l'application. Elle lui permet, par exemple, d'afficher le contenu de la liste lorsqu'une action ne se réalise pas comme il le faut. La méthode s'écrit comme suit :

```
public void afficherLesFormes() {
    int nbFormes = listeFormes.size();
    System.out.println("--------------------------------");
    if (nbFormes > 0) {
      // Pour toutes les formes de la liste, afficher son
      // contenu. La méthode afficher() utilisée est soit celle
      // de la classe Cercle, soit celle de la classe Rectangle,
      // selon le type de la forme.
      for (Forme tmp : listeFormes) tmp.afficher();
    }
    else {
      System.out.println("La liste des formes est vide");
    }
 }
```

- Enregistrer la liste des formes dans un fichier texte. La sauvegarde des données est réalisée par la méthode `enregistrerLesFormes()`. La liste des formes est enregistrée dans un fichier nommé `Formes.txt` qui possède autant de lignes qu'il y a d'objets placés dans la liste. Chaque forme est représentée par une chaîne de caractères spécifique :
  - un cercle est représenté par la chaîne `C;couleur;x;y;rayon ;`
  - un rectangle par la chaîne `R;couleur;x;y;largeur;hauteur`.

Copyright © 2016 Eyrolles.

- Extraire la liste des formes depuis un fichier texte. L'extraction des données et la mise sous forme de liste est réalisée par la méthode `lireLesFormes()`. Pour chaque ligne du fichier `Formes.txt`, la méthode extrait le premier caractère et crée un cercle ou un rectangle selon sa valeur. L'objet créé est ensuite ajouté à la liste `listeFormes`.

Extension Web Pour plus de précisions sur les méthodes `enregistrerLesFormes()` et `lireLesFormes()`, reportez-vous au fichier `Corriges.pdf` situé sur l'extension Web de cet ouvrage, section « Exercices 10-2 et 10-5 : Comprendre les listes et créer des fichiers textes ».

La classe FeuilleDeDessins

Une fois la structure des données mise en place, examinons comment réaliser l'affichage des données associées sur la feuille de dessins.

La feuille de dessins s'ajoute au projet `EditeurExemple` en cliquant droit sur l'item `EditeurExemple` de la fenêtre Projets et en sélectionnant les items Nouveau puis JPanel Form. De cette façon, la feuille de dessin, décrite par la classe `FeuilleDeDessins`, hérite des comportements de la classe `JPanel`.

En effet, comme nous avons pu l'observer au chapitre 11, « Dessiner des objets », un `JPanel` est un panneau de contenus. Il contient toutes les méthodes de base pour récupérer, par exemple, la position de la souris ou encore afficher un contenu graphique.

La classe `FeuilleDeDessins` et son constructeur s'écrivent, dans la fenêtre Source, comme suit :

```
public class FeuilleDeDessins extends javax.swing.JPanel {
  // La propriété principale de la classe : la liste d'affichage
  // listeAdessiner
  private ListeDeFormes listeAdessiner;
  // Le constructeur prend en paramètre la liste des formes créée
  // par l'application Main
  public FeuilleDeDessins (ListeDeFormes ldf) {
     // Initialiser la liste d'affichage à la liste des objets
     // passée en paramètre du constructeur
     listeAdessiner = ldf;
     // Afficher les composants et leurs gestionnaires
     // d'événements créés dans le panneau Design
     initComponents();
     // Rendre visible la feuille de dessins
     setVisible(true);
     // Donner une position et une taille à la feuille de dessins
     setBounds(0, 0,500, 500);
     // Modifier le curseur de la souris lorsqu'il se trouve sur
```

Copyright © 2016 Eyrolles.

```
        // la feuille de dessins
        setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
    }
}
```

La définition des comportements et des actions à mener par la feuille de dessins est décrite ci-après.

Définir le comportement des objets graphiques

Plusieurs comportements sont à définir pour la feuille de dessins. Le premier consiste à décrire la façon dont la feuille de dessins doit « se peindre ». Les autres comportements sont ceux liés aux actions de l'utilisateur avec la souris.

Qu'entend-t-on par « peindre » une feuille de dessins ?

Une fenêtre, et en particulier la feuille de dessins de l'éditeur, voit son contenu se modifier en fonction des actions de l'utilisateur. Ainsi, le contenu de la feuille de dessins change lorsque l'utilisateur dessine une nouvelle forme ou s'il décide d'effacer son dessin.

Lorsqu'une nouvelle fenêtre s'ouvre et se place sur l'éditeur graphique, le contenu de la feuille de dessins est effacé. Lorsque la fenêtre se ferme pour laisser réapparaître l'éditeur graphique, le contenu de la feuille de dessins doit à nouveau être réaffiché et la fenêtre doit être repeinte.

Pour afficher (peindre) la feuille de dessins, il convient « d'expliquer » à cette dernière comment le faire. Cette « explication » est réalisée grâce à la méthode `paintComponent()`.

La méthode `paintComponent()` est appelée à chaque fois que cela est nécessaire, soit par le système lorsque la fenêtre a été cachée et réapparaît, soit lorsque l'utilisateur crée de nouvelles formes. Dans ce dernier cas, la méthode qui dessine la forme souhaitée (par exemple, un cercle) fait appel à la fonction `repaint()`, qui elle-même appelle la méthode `paintComponent()`.

Examinons plus attentivement la méthode `paintComponent()` :

```
public void paintComponent (Graphics g) {
  // L'appel à la méthode paintComponent() de la classe mère (super)
  // permet l'affichage des éventuels autres composants placés
  // dans le panneau de contenu
  super.paintComponent(g);
  // Récupérer toutes les formes placées dans la listeAdessiner
  ArrayList<Forme> tmpListe = listeAdessiner.getListeFormes();
  int nbFormes = tmpListe.size();
  if (nbFormes > 0) {
   // Pour chaque forme contenue dans la liste,
   for (Forme f : tmpListe) {
    // dessiner la forme en utilisant la méthode dessiner() de
    // la classe Cercle ou Rectangle, selon son type
```

Copyright © 2016 Eyrolles.

```
        f.dessiner(g);
        }
      }  else {
        System.out.print("Il n'y a pas de forme dans cette liste");
      }
}
```

Remarque La méthode `dessiner()` est définie à deux reprises : une fois dans la classe `Cercle` et une seconde fois dans la classe `Rectangle`. Les instructions qui composent ces deux méthodes diffèrent puisque le dessin d'un rectangle est réalisé par la méthode `fillRect()` alors que celui d'un cercle est effectué par la méthode `fillOval()`.

Dessiner sur un clic

Une forme se dessine à la souris : l'utilisateur clique sur la feuille à l'endroit où il souhaite commencer son dessin et lorsqu'il relâche le bouton de la souris, le tracé de la forme apparaît.

Pour dessiner une forme, il suffit donc d'enregistrer les coordonnées de la souris au moment du clic et au moment du relâchement du bouton de la souris.

Ces deux opérations sont réalisées par les deux gestionnaires d'événements suivants :

```
// Détecter lorsque l'utilisateur clique sur la feuille de dessins
private void formMousePressed(java.awt.event.MouseEvent evt) {
   // Enregistrer les coordonnées de la souris au moment du clic
   debutX = evt.getX();
   debutY = evt.getY();
}
// Détecter lorsque l'utilisateur relâche le bouton de la souris
private void formMouseReleased(java.awt.event.MouseEvent evt) {
   // Enregistrer la distance parcourue par la souris, en X et
   // en Y, entre le moment du clic et celui du relâchement
   deltaX = evt.getX()- debutX;
   deltaY = evt.getY()- debutY;
   // Si la forme est un cercle, le dessiner
   if (forme.equals("cercle")) {
       dessinerUnCercle();
   }
   // Si la forme est un rectangle, le dessiner
   if (forme.equals("rectangle")) {
       dessinerUnRectangle();
   }
}
```

Copyright © 2016 Eyrolles.

Les variables `debutX`, `debutY`, `deltaX`, `deltaY` et `forme` sont définies comme propriétés de la classe `FeuilleDeDessins`. Elles sont déclarées en dehors de toutes fonctions, comme suit :

```
private int debutX, debutY, deltaX, deltaY;
private String forme="cercle";
```

La variable `forme` est initialisée à `"cercle"`. De cette façon, l'utilisateur dessine par défaut des cercles, à moins qu'il ne clique sur l'outil Rectangle. Dans ce cas, la propriété `forme` prend la valeur `"rectangle"` et le dessin de rectangles remplace celui de cercles. Le changement de la valeur `forme` est réalisé par les gestionnaires d'événements des boutons `cercle` et `rectangle` (voir la section suivante « Dessiner la bonne forme »).

Dessiner un cercle à la souris

Il existe quatre façons de tracer un cercle à la souris. Chacune de ces méthodes est représentée sur la figure 12-38.

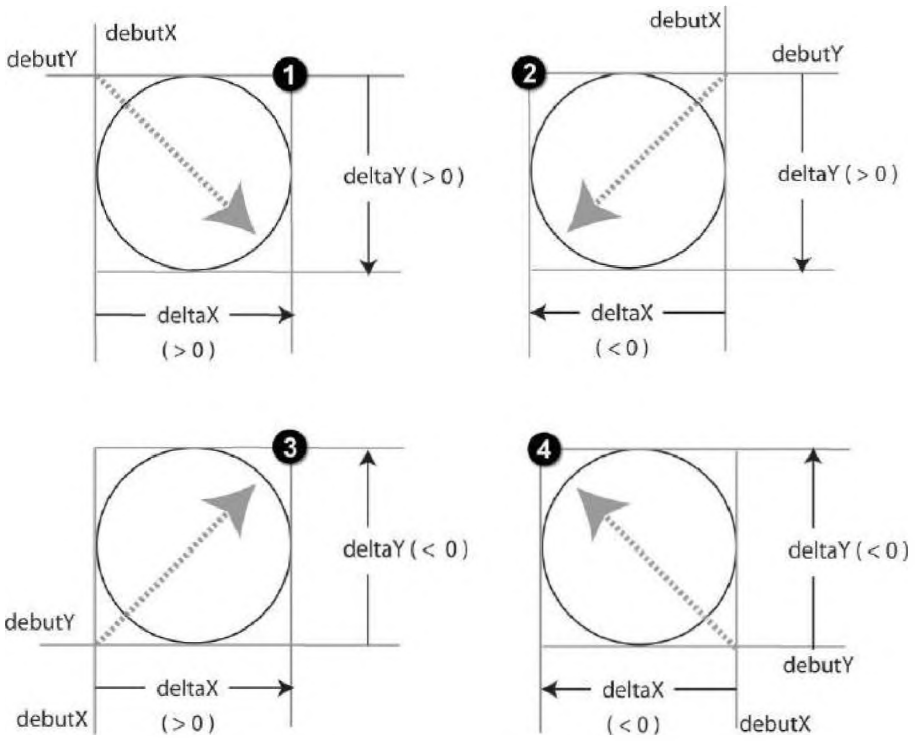


Figure 12-38 *Les différentes façons de dessiner un cercle (voir les puces ❶, ❷, ❸ et ❹ du code ci-après)*

❶ La première méthode, la plus classique, consiste à cliquer sur la feuille, puis à déplacer le curseur de la souris vers la droite et vers le bas. Il s'agit du cas le plus simple, puisque la fonction `fillOval()` demande en paramètres les coordonnées qui correspondent

Copyright © 2016 Eyrolles.

exactement à celles de la boîte (carrée) qui englobe le cercle à tracer, c'est-à-dire les valeurs `debutX`, `debutY`, `deltaX` ou `deltaY`.

Remarque

L'éditeur graphique propose de dessiner des cercles et non des ellipses. La boîte « englobante » est donc nécessairement carrée. La valeur prise par la hauteur et la largeur de la boîte doit être `deltaX` ou `deltaY`. La valeur de `deltaX` est choisie lorsqu'elle est plus grande que celle de `deltaY`. La valeur ainsi choisie est stockée dans une variable locale nommée `delta` (voir le code source de la méthode `dessinerUnCercle()` ci-après).

❷ Avec la deuxième méthode, le tracé du cercle s'effectue de droite à gauche et de haut en bas. La valeur `deltaX` est négative et correspond au diamètre du cercle à tracer. Nous devons la rendre positive. Il est également nécessaire de faire en sorte que le point initial du tracé corresponde au coin supérieur gauche de la boîte englobante. Pour cela, nous utilisons une variable nommée `decalageX` afin de déplacer l'origine de la boîte englobante au coin supérieur gauche du tracé. Ce décalage vaut ici `-deltaX`.

❸ Ici, le tracé du cercle est réalisé de gauche à droite et de bas en haut. La valeur `deltaY` est négative, nous devons de la même façon la rendre positive et faire en sorte que le point initial du tracé corresponde au coin supérieur gauche de la boîte englobante. Pour cela, nous utilisons une variable nommée `decalageY` afin de déplacer l'origine de la boîte englobante au coin supérieur gauche du tracé. Ce décalage vaut ici `-deltaY`.

❹ Pour finir, il est aussi possible de tracer un cercle de droite à gauche et de bas en haut. Selon que `deltaX` est plus grand ou plus petit que `deltaY`, le déplacement de la boîte englobante vers le coin supérieur gauche du tracé s'effectue soit sur l'axe des X, soit sur l'axe des Y.

La méthode `dessinerUnCercle()` ci-après décrit en langage Java comment tracer un cercle quelle que soit la méthode de sélection utilisée.

```
private void dessinerUnCercle() {
   int delta=0;
   int decalageX = 0;
   int decalageY = 0;
   // Si la largeur de la boîte englobante est supérieure
   // à sa hauteur,
   if (Math.abs(deltaX) > Math.abs(deltaY) ) {
       // ❶ stocker la largeur de la boîte englobante
       delta = deltaX;
       // Si le tracé de la boîte a été effectué de droite
       // à gauche,
       if (delta < 0) {
           // ❷ placer l'origine de la boîte englobante au coin
           // supérieur gauche du tracé
           decalageX = delta;
```

Copyright © 2016 Eyrolles.

```
                // et rendre positif le diamètre
                delta = -delta;
            }
            // Si le tracé de la boîte a été effectué de gauche
            // à droite, il n'y a pas de décalage
            else decalageX = 0;
        }
        // Si la hauteur de la boîte englobante est supérieure
        // à sa largeur,
        if (Math.abs(deltaY) > Math.abs(deltaX) ) {
            // ❸ stocker la hauteur de la boîte englobante
            delta = deltaY;
            // Si le tracé de la boîte a été effectué de bas en haut,
            if (delta < 0) {
                // ❹ placer l'origine de la boîte englobante au
                //    coin supérieur gauche du tracé
                decalageY = delta;
                delta = -delta;
            }
            // Si le tracé de la boîte a été effectué de haut en bas,
            // il n'y a pas de décalage
            else decalageY = 0;
        }
        // ❺ Créer un cercle avec les valeurs de saisie
        Cercle c = new Cercle(debutX + decalageX,
                              debutY + decalageY, delta, couleur );
        // ❻ Ajouter le cercle à la liste d'affichage
        listeAdessiner.ajouterUneForme(c);
        // ❼ Repeindre la fenêtre qui a pour effet d'appeler
        //    la méthode paintComponent()
        repaint();
    }
```

❺ À l'issue des tests, les valeurs `decalageX`, `decalageY`, et `delta` sont initialisées aux valeurs qui correspondent au mode de tracer du cercle. Un cercle est créé à l'aide du constructeur de la classe `Cercle` avec en paramètres la position et la taille qui lui conviennent.

❻ L'objet créé est ensuite ajouté à la liste d'affichage `listeAdessiner`.

❼ Pour finir, l'objet est dessiné sur la feuille par l'intermédiaire de la méthode `repaint()` qui fait appel à la méthode `paintComponent()` décrite à la section précédente « Qu'entend-t-on par « peindre » une feuille de dessins ? ».

Copyright © 2016 Eyrolles.

Relier la feuille de dessins à la fenêtre principale

La feuille de dessins est créée par l'application `Main` comme suit :

```
public class Main extends javax.swing.JFrame {
   // Définition des propriétés :
   // - la liste d'affichage liste
   private ListeDeFormes liste;
   // ❶ - la feuille de dessins page
   private FeuilleDeDessins page;
   // Constructeur de la classe Main()
   public Main() {
       // Définition de la taille de la fenêtre de l'éditeur
       setBounds(100, 100,600, 600);
       // Afficher les composants et leurs gestionnaires
       // d'événements créés dans le panneau Design
       initComponents();
       // Créer une liste d'affichage vide
       liste = new ListeDeFormes();
       // ❷ Créer une feuille de dessins
       page = new FeuilleDeDessins(liste);
       // ❸ Insérer la page dans le composant boiteDessin créé
       //dans le panneau Design
       boiteDessin.add(page);
    }
    // Définitions de la boîte à outils et des gestionnaires
    // d'événements, voir section ci-après
}
```

❶ La feuille de dessins est une propriété de la classe `Main`, nommée `page`.

❷ L'objet `page` est créé par le constructeur de la classe `Main` en prenant en paramètre la liste des objets à afficher. Au lancement de l'application, la liste est vide, la feuille de dessins reste blanche.

❸ La feuille de dessins est placée sur le panneau d'affichage de l'application principale grâce au composant `boiteDessin`. Ce composant est un `JPanel` que nous avons pris soin de placer sous la boîte à outils, par l'intermédiaire du panneau Design.

Remarque Il est nécessaire de créer la liste d'affichage au sein de la classe `Main` car même si les objets sont créés par la classe `FeuilleDeDessins`, ils peuvent être supprimés de la liste avec des outils (Effacer, Créer une nouvelle feuille, etc.) définis dans la classe `Main`. La liste d'affichage est « connue » de la feuille de dessins lorsqu'elle est passée en paramètre du constructeur `FeuilleDeDessins()`.

Copyright © 2016 Eyrolles.

Créer une boîte à outils

La boîte à outils propose, comme son nom l'indique, un ensemble d'outils pour tracer des formes circulaires ou rectangulaires, choisir une couleur ou encore effacer tout ou partie de la feuille de dessins.

Mise en place des éléments graphiques

Les éléments de la boîte à outils sont regroupés par thème au sein d'un composant de type `JToolBar`.

La mise en place d'une `JToolBar` se fait par un simple glisser-déposer du composant, depuis le panneau Palette vers le panneau Design. Le composant est placé au-dessus du composant `boiteDessins` et nommé `boiteOutils`.

La boîte à outils est composée à son tour de composants de type bouton, séparateur ou encore bouton à bascule. L'ajout d'un composant au sein de `boiteOutils` se fait tout aussi simplement, par un simple glisser-déposer du composant, depuis le panneau Palette vers la `JToolBar`.

Nous détaillons ci-après les composants de la `boiteOutils`, par thème.

Le thème « Choisir une forme »

La création du thème « Choisir une forme » passe par l'utilisation des composants `JToggleButton` et `ButtonGroup`.

En effet, le choix de la forme à dessiner s'effectue en cliquant sur l'outil Cercle ou sur l'outil Rectangle. Lorsque l'utilisateur sélectionne l'un de ces deux outils, le bouton cliqué reste sélectionné et visiblement enfoncé tant que l'autre outil n'est pas sélectionné.

Dans le cas de la sélection d'un outil, l'emploi du `JToggleButton` est utile pour montrer à l'utilisateur quel est l'outil de tracé en cours de fonctionnement. En effet, le composant `JToggleButton` ne réagit pas comme un simple bouton mais comme un bouton à bascule, un bouton à deux états. Pour passer d'un état à un autre, l'utilisateur doit cliquer dessus.

Pour notre exemple, nous utilisons deux `JToggleButton` nommés `cercle` et `rectangle` qui sont programmés de sorte que lorsque `cercle` est sélectionné, `rectangle` ne l'est pas.

La désélection d'un outil est réalisée par la sélection de l'autre outil. Ces deux actions sont traitées par l'utilisation du composant `ButtonGroup`.

Le composant `ButtonGroup` est un composant non graphique qui va, lorsqu'on le place sur la scène de la fenêtre Design, se ranger directement dans la hiérarchie Other Components de la fenêtre Inspecteur (voir figure 12-39). Nommons `groupeForme` le groupe des boutons associés au thème « Choisir une forme ».

La mise en place d'un ensemble de boutons au sein d'un même `ButtonGroup` signifie qu'un seul des boutons du groupe peut être sélectionné à la fois. Ainsi, placer les boutons `cercle` et `rectangle` au sein du groupe `groupeForme` à pour conséquence que seule une des deux formes peut être sélectionnée à la fois.

Copyright © 2016 Eyrolles.

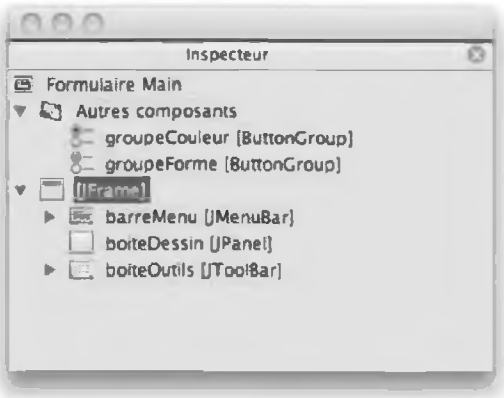


Figure 12-39 *Le panneau Inspector de la classe Main*

L'ajout des boutons `cercle` et `rectangle` au sein du groupe `groupeForme` s'écrit :

```
// Ajouter le bouton cercle au groupe groupeForme
groupeForme.add(cercle);
// Ajouter le bouton rectangle au groupe groupeForme
groupeForme.add(rectangle);
// Le bouton cercle est affiché comme sélectionné
cercle.setSelected(true);
```

Ensuite, les boutons `cercle` et `rectangle` sont affichés sous la forme d'icônes. Un dessin remplace le texte qu'ils contiennent. Ce dessin est enregistré dans un fichier au format PNG.

L'insertion d'une image dans un composant `JToggleButton` est réalisée en créant un objet de type `ImageIcon` grâce aux instructions :

```
ImageIcon iconPhoto = new ImageIcon(url+"cercle.png");
cercle.setIcon(iconPhoto);
```

Pour en savoir plus L'`url` est initialisé à `"Ressources/"`. Le répertoire `Ressources` est créé dans le répertoire associé au projet `EditeurExemple`. L'organisation du système de fichiers créé par NetBeans est décrite dans l'annexe « Guide d'installations », section « Utilisation des outils de développement ».

Pour simplifier la mise en place du groupe de boutons « Choisir une forme », nous insérons l'ensemble des instructions qui le crée dans la fonction `creerThemeForme()` comme suit :

```
private void creerThemeForme() {
    // Insérer les cercle et rectangle dans le groupe de boutons
    // groupeForme
    groupeForme.add(cercle);
```

Copyright © 2016 Eyrolles.

```
        groupeForme.add(rectangle);
        // L'outil cercle est sélectionné par défaut
        cercle.setSelected(true);
        // Placer l'image cercle.png sur le bouton cercle
        ImageIcon iconPhoto = new ImageIcon(url+"cercle.png");
        // Afficher une infobulle au survol du bouton cercle
        cercle.setToolTipText("Dessiner des cercles");
        cercle.setIcon(iconPhoto);
        // Placer l'image rectangle.png sur le bouton rectangle
        iconPhoto = new ImageIcon(url+"rectangle.png");
        // Afficher une infobulle au survol du bouton rectangle
        rectangle.setToolTipText("Dessiner des rectangles");
        rectangle.setIcon(iconPhoto);
    }
```

La fonction est ensuite appelée par le constructeur de la classe `Main`.

Remarque

La méthode `setToolTipText()` est utilisée pour afficher automatiquement une infobulle au survol du composant sur lequel est appliqué la méthode. Ici, par exemple, au survol de l'icône représentant un cercle, une infobulle contenant le texte « Dessiner des cercles » s'affichera.

Le thème « Choisir une couleur »

La gestion des outils de coloriage s'écrit de la même façon que pour les outils Cercle et Rectangle. La fonction `creerThemeForme()` qui affiche les cinq choix de couleur sous forme de boutons, s'écrit comme suit :

```
private void creerThemeCouleur() {
    // Insérer les boutons rouge, vert, bleu, etc., dans le groupe
    // de boutons groupeCouleur
    groupeCouleur.add(rouge);
    groupeCouleur.add(vert);
    groupeCouleur.add(bleu);
    groupeCouleur.add(blanc);
    groupeCouleur.add(noir);
    // La couleur noire est sélectionnée par défaut
    noir.setSelected(true);
    // Placer l'image rouge.png sur le bouton rouge
    ImageIcon iconPhoto = new ImageIcon(url+"rouge.png");
    rouge.setIcon(iconPhoto);
    rouge.setToolTipText("Peindre en rouge");
    // Placer l'image vert.png sur le bouton vert
    iconPhoto = new ImageIcon(url+"vert.png");
```

Copyright © 2016 Eyrolles.

```
    vert.setIcon(iconPhoto);
    vert.setToolTipText("Peindre en vert");
    // Placer l'image bleu.png sur le bouton bleu
    iconPhoto = new ImageIcon(url+"bleu.png");
    bleu.setToolTipText("Peindre en bleu");
    bleu.setIcon(iconPhoto);
    // Placer l'image blanc.png sur le bouton blanc
    iconPhoto = new ImageIcon(url+"blanc.png");
    blanc.setToolTipText("Peindre en blanc");
    blanc.setIcon(iconPhoto);
    // Placer l'image noir.png sur le bouton noir
    iconPhoto = new ImageIcon(url+"noir.png");
    noir.setToolTipText("Peindre en noir");
    noir.setIcon(iconPhoto);
}
```

Le thème « Effacer »

Les outils qui permettent d'effacer l'intégralité de la feuille de dessins ou la dernière forme tracée sont de simples boutons, indépendants les uns des autres. Il est inutile de créer un groupe de boutons. Leur affichage est réalisé par la fonction `creerThemeEffacer()` suivante :

```
private void creerThemeEffacer() {
    // Placer l'image poubelle.png sur le bouton effacer
    ImageIcon iconPhoto = new ImageIcon(url+"poubelle.png");
    // Afficher une infobulle au survol du bouton effacer
    effacer.setToolTipText("Tout effacer");
    effacer.setIcon(iconPhoto);
    // Placer l'image undo.png sur le bouton undo
    iconPhoto = new ImageIcon(url+"undo.png");
    // Afficher une infobulle au survol du bouton undo
    undo.setToolTipText("Effacer le dernier");
    undo.setIcon(iconPhoto);
}
```

Définir le comportement des objets graphiques

Une fois affichée la barre d'outils et les boutons qui la composent, il nous reste à décrire les actions menées par chacun d'entre eux.

Dessiner la bonne forme

Cliquer sur le bouton `cercle` ou `rectangle` revient à sélectionner la forme à dessiner. Pour cela, nous devons indiquer à l'objet `page` quelle forme doit être tracée, en fonction du bouton sélectionné.

Copyright © 2016 Eyrolles.

Ces actions sont décrites par les gestionnaires d'événements associés aux boutons `cercle` ou `rectangle`, comme suit :

```
private void rectangleActionPerformed(java.awt.event.ActionEvent
                                      evt) {
  // Initialiser la propriété forme de la classe FeuilleDeDessins
  // à "rectangle"
  page.setForme("rectangle");
}

private void cercleActionPerformed(java.awt.event.ActionEvent evt) {
  // Initialiser la propriété forme de la classe FeuilleDeDessins
  // à "cercle"
  page.setForme("cercle");
}
```

Les deux gestionnaires `rectangleActionPerformed()` et `rectangleActionPerformed()` initialisent la propriété `forme` de l'objet `page` à la valeur (`"cercle"` ou `"rectangle"`) qui lui correspond. Cette initialisation est réalisée par la méthode d'accès en écriture `setForme()` définie dans la classe `FeuilleDeDessins`.

De cette façon, lorsque l'utilisateur clique sur la feuille de dessins, le gestionnaire `formMouseReleased()` « sait », en testant le contenu de la propriété `forme`, quelle forme doit être dessinée (voir la section précédente « Dessiner sur un clic »).

Remarque

Par défaut, le bouton `cercle` est sélectionné. Au lancement de l'application, l'utilisateur trace des cercles, s'il ne clique pas sur le bouton `rectangle`. La propriété `forme` doit être initialisée dès sa création à `"cercle"` dans la classe `FeuilleDeDessins`.

Colorier avec la bonne couleur

Lorsque l'utilisateur souhaite tracer une forme rouge, il clique sur le bouton de peinture rouge. S'il veut changer de couleur, il clique sur le bouton de la couleur de son choix.

Le choix de la couleur de coloriage est donc réalisé par les gestionnaires d'événements associés aux boutons `rouge`, `vert`, `bleu`, `blanc` et `noir`, comme suit :

```
private void rougeActionPerformed(java.awt.event.ActionEvent evt) {
  // Initialiser la propriété couleur de la classe FeuilleDeDessins
  // à Forme.ROUGE
  page.setCouleur(Forme.ROUGE);
}
```

Copyright © 2016 Eyrolles.

```
private void vertActionPerformed(java.awt.event.ActionEvent evt) {
  // Initialiser la propriété couleur de la classe FeuilleDeDessins
  // à Forme.VERT
  page.setCouleur(Forme.VERT);
}
private void bleuActionPerformed(java.awt.event.ActionEvent evt) {
  // Initialiser la propriété couleur de la classe FeuilleDeDessins
  // à Forme.BLEU
  page.setCouleur(Forme.BLEU);
}
private void blancActionPerformed(java.awt.event.ActionEvent evt) {
  // Initialiser la propriété couleur de la classe FeuilleDeDessins
  // à Forme.BLANC
  page.setCouleur(Forme.BLANC);
}
private void noirActionPerformed(java.awt.event.ActionEvent evt) {
  // Initialiser la propriété couleur de la classe FeuilleDeDessins
  // à Forme.NOIR
  page.setCouleur(Forme.NOIR);
}
```

Les gestionnaires associés aux boutons de coloriage initialisent la propriété `couleur` de l'objet `page` à la valeur (`Forme.ROUGE`, `Forme.BLEU`, etc.) qui lui correspond. Cette initialisation est réalisée par la méthode d'accès en écriture `setCouleur()`, définie dans la classe `FeuilleDeDessins`.

Ensuite, la couleur d'affichage est transmise à l'objet `Cercle` ou `Rectangle` créé, par l'intermédiaire de son constructeur (voir la section précédente « Dessiner un cercle à la souris »).

Par défaut, le bouton de couleur noire est sélectionné. Au lancement de l'application, l'utilisateur trace des cercles de couleur noire, s'il ne sélectionne pas une autre couleur. La propriété `couleur` est donc initialisée, dès sa création, à `Forme.Noir` dans la classe `FeuilleDeDessins`.

Remarque

Les constantes `Forme.NOIR`, `Forme.ROUGE`, etc., sont définies comme variables statiques dans la classe `Forme`. Ce sont des valeurs numériques qui, pour chacune, correspondent à l'indice de la couleur associée, définie dans le tableau `couleurDessin`. Ce dernier est lui-même également déclaré comme propriété statique de la classe `Forme`.

Effacer la page

Pour effacer les formes tracées sur la feuille de dessins, l'utilisateur clique sur le bouton représentant une poubelle, le bouton `effacer`.

Copyright © 2016 Eyrolles.

La suppression des objets de la feuille de dessins est réalisée par le gestionnaire d'événements associé au bouton `effacer` suivant :

```
private void effacerActionPerformed(java.awt.event.ActionEvent evt)
{
  liste.supprimerLesFormes();
  page.dessinerLesFormes(liste);
}
```

L'effacement du dessin s'effectue en deux temps :

- Supprimer les objets de la `liste` d'affichage, grâce à la méthode `supprimerLesFormes()` de la classe `ListeDeFormes`.
- Repeindre la `page` avec une liste d'affichage vide, par simple appel à la méthode `dessinerLesFormes()` de la classe `FeuilleDeDessins`.

Pour en savoir plus L'opération qui consiste à supprimer la dernière forme dessinée (undo) est traitée en exercice (voir la section « Exercices - L'éditeur graphique version 2 » à la fin de ce chapitre).

Créer un menu

Le menu se décompose en deux items :

- l'item Fichier qui propose très classiquement de créer une nouvelle feuille de dessins, d'ouvrir une feuille existante, d'enregistrer un dessin et de quitter l'application ;
- l'item Aide qui est utilisé pour afficher une fenêtre À propos.

Mise en place des éléments graphiques

Les items du menu sont regroupés au sein d'un composant de type `JMenuBar`.

La mise en place d'une `JMenuBar` se fait par un simple glisser-déposer du composant, depuis le panneau Palette vers le panneau Design. Le composant est placé par NetBeans à l'origine de la fenêtre d'application. Nous nommons ce composant `barreMenu`.

Par défaut, la `JMenuBar` est composée de deux `JMenu` avec comme noms d'item File et Edit. Pour modifier ces noms, il suffit de double-cliquer lentement sur les `JMenu` et d'en modifier les champs de texte. Nous vous proposons de remplacer le terme File par « Fichier » et Edit par « Aide ».

Les composants JMenuItem

Le menu Fichier est constitué de composants de type `JMenuItem`. L'ajout d'un composant au sein de l'objet `barreMenu` se fait par un simple glisser-déposer du composant, depuis le panneau Palette vers le `JMenu` souhaité (voir figure 12-40-❶).

Copyright © 2016 Eyrolles.

Figure 12-40 *Créer les items d'une barre de menus*

Nous modifions ensuite le texte du nouvel item, en double-cliquant lentement dessus (voir figure 12-40-❷).

Il est aussi possible de définir des raccourcis clavier pour chaque item du menu. Cette fonctionnalité se réalise très simplement en double-cliquant sur le terme « shortcut » situé juste à côté du nom de l'item, ce qui a pour effet d'ouvrir la fenêtre représentée à la figure 12-41.

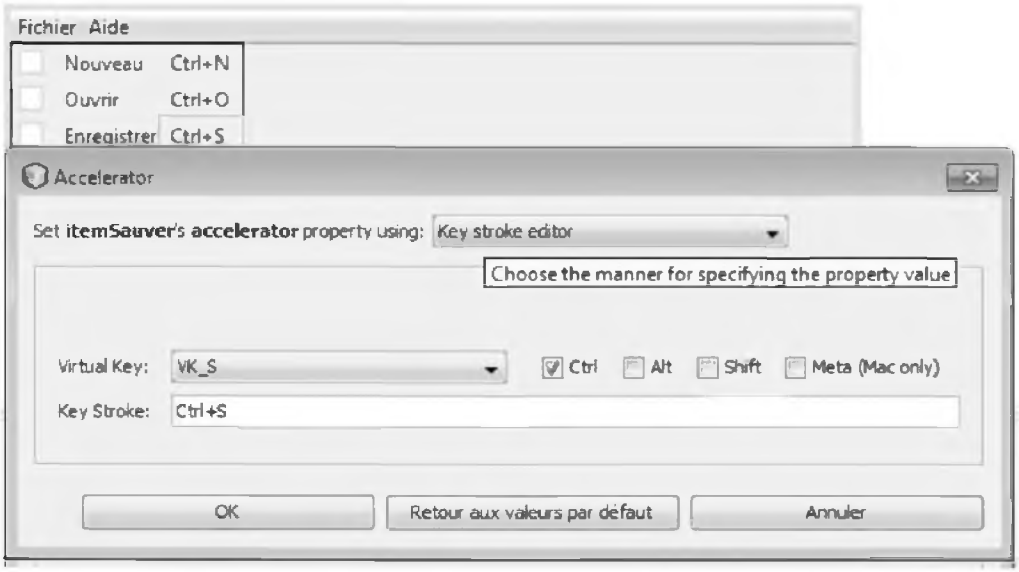


Figure 12-41 *La fenêtre Accelerator*

L'ajout du raccourci se fait en sélectionnant le champ Key Stroke, puis en appuyant sur les touches du clavier que l'on souhaite utiliser comme raccourci. Après validation via le bouton OK, le raccourci clavier est automatiquement relié au gestionnaire d'événements associé à l'item du menu sélectionné.

Copyright © 2016 Eyrolles.

Définir le comportement des objets graphiques

Une fois affichée la barre de menus et les items qui la composent, nous devons décrire les actions menées par chacun d'entre eux.

Créer une nouvelle page

La création d'une nouvelle page est réalisée par le gestionnaire de l'item Nouveau. Les instructions qui le composent sont :

```
private void itemNouveauActionPerformed(java.awt.event.ActionEvent
                                                    evt) {
  if (liste != null) liste.supprimerLesFormes();
  page.dessinerLesFormes(liste);
}
```

Créer une nouvelle feuille de dessins revient à effacer le contenu de la `liste` des formes en cours puis à repeindre la `page` avec une liste d'affichage vide, par simple appel à la méthode `dessinerLesFormes()` de la classe `FeuilleDeDessins`.

Si la `liste` est vide, il n'est pas nécessaire d'en supprimer les éléments puisqu'il n'y en a pas.

Enregistrer un dessin

L'enregistrement sous forme de fichier texte de votre dessin est réalisé par le gestionnaire de l'item Enregistrer. Les instructions qui le composent sont :

```
private void itemSauverActionPerformed(java.awt.event.ActionEvent
                                                    evt) {
  Fichier f = new Fichier();
  f.ouvrir("Formes.txt", "W");
  if (liste != null)  liste.enregistrerLesFormes(f);
  f.fermer();
}
```

Le gestionnaire `itemSauverActionPerformed()` utilise la classe `Fichier` développée au chapitre 10, « Collectionner un nombre indéterminé d'objets », section « Exercices – Créer des fichier texte ».

À l'appel du gestionnaire associé à l'objet `itemSauver`, un objet de type `Fichier` est créé puis ouvert en écriture. Le fichier a obligatoirement pour nom : `"Formes.txt"`.

Pour en savoir plus L'opération qui consiste à « Enregistrer sous... » et permet à l'utilisateur de donner le nom qu'il souhaite à son dessin est traitée en exercice (voir la section « Exercices – L'éditeur graphique version 2 » à la fin de ce chapitre).

Copyright © 2016 Eyrolles.

Ensuite, si la liste d'affichage n'est pas vide, le contenu de la `liste` est enregistré dans le fichier texte par l'intermédiaire de la méthode `enregistrerLesFormes()`.

Vous pouvez consulter son contenu en éditant le fichier `Formes.txt` à l'aide de Bloc-notes ou TextEdit. Le fichier est enregistré dans le répertoire `EditeurExemple` du système de fichiers créé par NetBeans.

Pour en savoir plus L'organisation du système de fichiers créé par NetBeans est décrite dans l'annexe « Guide d'installation », section « Utilisation des outils de développement ».

Ouvrir une page dessinée

Lorsque l'utilisateur sélectionne l'item Ouvrir, les formes enregistrées dans le fichier `Formes.txt` sont tracées sur la feuille de dessins. Elles remplacent le dessin éventuellement présent sur la feuille.

L'ouverture du fichier `Formes.txt` est réalisée par le gestionnaire de l'item Ouvrir. Les instructions qui le composent sont les suivantes :

```
private void itemOuvrirActionPerformed(java.awt.event.ActionEvent
                                                    evt) {
  if (liste != null) liste.supprimerLesFormes();
  Fichier f = new Fichier();
  if (f.ouvrir("Formes.txt", "R")) {
      liste.lireLesFormes(f);
      f.fermer();
      page.dessinerLesFormes(liste);
  }
}
```

Si la liste d'affichage en cours de traitement n'est pas vide, en d'autres termes, si un dessin est présent sur la feuille de dessins, on supprime les objets présents dans la liste, ce qui a pour conséquence d'effacer le dessin éventuellement présent.

Le fichier `Formes.txt` est ensuite ouvert en lecture afin d'en extraire les formes et de les enregistrer dans l'objet `liste`. C'est ce que réalise la méthode `lireLesFormes()`.

Une fois la `liste` remplie des formes enregistrées dans le fichier texte, le dessin associé est affiché sur la page par l'intermédiaire de la méthode `dessinerLesFormes()`.

Quitter l'application

Pour quitter l'application, l'utilisateur sélectionne l'item Quitter ou utilise le raccourci clavier `Ctrl + Q`. Le gestionnaire d'événements associé à l'objet `itemQuitter` s'écrit tout simplement comme suit :

Copyright © 2016 Eyrolles.

```
private void itemQuitterActionPerformed(java.awt.event.ActionEvent
evt) {
  System.exit(0);
}
```

Résumé

L'utilisation d'un EDI tel que NetBeans simplifie grandement la conception d'applications munies d'interfaces graphiques conviviales. NetBeans est un environnement de développement, développé par Sun et distribué en Open Source.

NetBeans propose un outil basé sur deux représentations d'une même application :

- la représentation graphique visible sur le panneau Design ;
- la représentation textuelle du code, visible dans le panneau Source.

Pour passer d'une représentation graphique à une représentation « codée », il suffit de cliquer sur l'onglet correspondant, à savoir Design ou Source.

La fabrication d'interfaces graphiques passe ensuite par deux étapes distinctes :

- La mise en place des éléments graphiques. Cette étape s'effectue très simplement en sélectionnant le composant de son choix dans la palette de composants basés sur la bibliothèque Swing. Il suffit de faire glisser le composant jusqu'au panneau Design.
- La définition du comportement des objets graphiques. Une fois placés et nommés, les composants prennent « vie » en insérant les instructions décrivant les actions à réaliser, au sein de fonctions spécifiques. En programmation événementielle, ces fonctions sont appelées des gestionnaires d'événements.
- Avec NetBeans, il suffit de double-cliquer sur le composant souhaité pour créer son gestionnaire d'événements. L'interface se place d'elle-même sur le panneau Source, à l'intérieur de la fonction correspondant au gestionnaire concerné.

Exercices

S'initier à NetBeans

L'objectif est ici de construire une application relativement simple qui convertit des mètres en centimètres, des litres en décilitres et des heures en secondes.

L'application se présente sous la forme suivante (voir figure 12-42) :

Copyright © 2016 Eyrolles.

Figure 12-42 Le convertisseur de mesure

Mise en place des éléments graphiques

Exercice 12.1

a. À l'aide de l'interface NetBeans, créez un projet nommé `Exercice12_1` auquel vous ajouterez ensuite une classe `Main` basée sur le composant `JFrame`.

b. Dans le panneau Design de la classe `Main`, placez les composants de façon à obtenir le visuel présenté à figure 12-42. Les composants à utiliser sont : `JRadioButton`, `JSeparator`, `JTextField`, `JLabel`, `JButton` et `ButtonGroup`.

c. Modifiez les noms des composants et leur contenu de façon à obtenir un panneau Inspecteur tel que celui présenté à la figure 12-43.

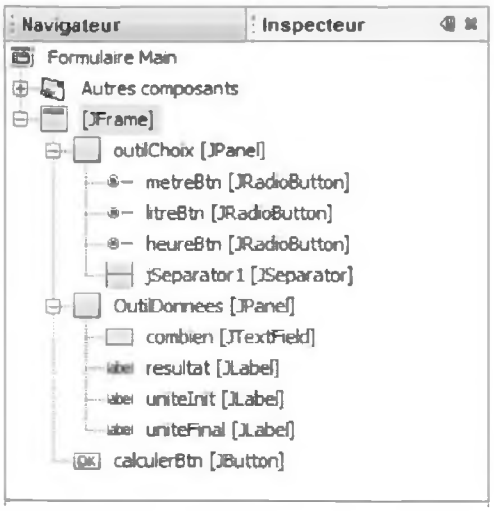


Figure 12-43 Le panneau Inspecteur associé au convertisseur

Copyright © 2016 Eyrolles.

d. Modifiez les champs `text` des composants de type `JLabel`, `JButton` et `JRadioButton` afin d'obtenir une fenêtre telle que celle présentée à la figure 12-42.

Définir le comportement des composants

Exercice 12.2 Le choix du type de mesure à convertir est réalisé par le groupe de boutons `groupeRadioBtn`.

a. Ajoutez les boutons `metreBtn`, `litreBtn` et `heureBtn` au groupe de boutons `groupeRadioBtn`.

b. Faites en sorte que le bouton radio `metreBtn` soit sélectionné par défaut.

Lorsque l'utilisateur sélectionne l'une des trois unités de mesure, le contenu textuel des labels `uniteFinal` et `uniteInit` est modifié. Ainsi, `"Mètre(s)"` est remplacé par `"Litre(s)"` ou `"Heure(s)"` et `"Centimètre(s)"` est remplacé par `"Décilitre(s)"` ou `"Seconde(s)"`.

c. Écrivez les gestionnaires d'événements des boutons `metreBtn`, `litreBtn` et `heureBtn` qui réalisent ces modifications.

d. Dans la classe `Main`, créez une propriété `choixAction` initialisée à la chaîne de caractères `"Metre"`. Faites en sorte que la propriété `choixAction` prenne la valeur :

- `"Litre"` quand l'utilisateur sélectionne le bouton `litreBtn` ;
- `"Heure"` quand l'utilisateur sélectionne le bouton `heureBtn`.

Exercice 12.3 La conversion d'une valeur est affichée lorsque l'utilisateur clique sur le bouton `calculerBtn`.

a. Ajoutez un gestionnaire d'événements au bouton `calculerBtn` dans lequel vous récupérerez la valeur saisie dans le champ de saisie `combien`.

b. Faites en sorte que la conversion soit calculée en fonction de l'unité de mesure choisie. Pour cela, écrivez trois fonctions `convertirMetre()`, `convertirHeure()` et `convertirLitre()` qui prennent en paramètre la valeur saisie et retournent en résultat une chaîne de caractères correspondant à la valeur convertie dans l'unité de mesure correspondante.

c. Affichez le résultat obtenu dans le label `resultat`.

Le gestionnaire d'étudiants version 2

L'objectif de cet exercice est d'améliorer le gestionnaire d'étudiants présenté au cours de ce chapitre.

Les options Créer, Modifier, Supprimer

L'application présente maintenant trois nouvelles options : Créer, Modifier et Supprimer (voir figure 12-44).

Copyright © 2016 Eyrolles.

Figure 12-44 *Les nouvelles options du gestionnaire d'étudiants*

Exercice 12.4 Les trois options Créer, Modifier et Supprimer se présentent sous la forme d'un groupe de trois boutons radio.

a. Dans l'interface NetBeans, reprennez le projet `GestionClasseExemple` développé au cours de ce chapitre. Faites-en une copie que vous nommerez `GestionClasseExercice`.

b. Modifiez la fenêtre Gestion des étudiants en y ajoutant les trois boutons radio que vous nommerez `creerRadioBtn`, `modifierRadioBtn` et `supprimerRadioBtn`. Ajoutez également un groupe de boutons que vous nommerez `groupeBtn`.

c. Ajoutez les boutons `creerRadioBtn`, `modifierRadioBtn` et `supprimerRadioBtn` au groupe de boutons `groupeBtn`. Faites en sorte que le bouton `creerRadioBtn` soit sélectionné par défaut.

d. Dans la classe `CursusSwing`, créez une propriété `choixAction` de type `String` et initialisez-la à `"Creer"`.

e. Écrivez les instructions qui font que la variable `choixAction` est initialisée respectivement à `"Creer"`, `"Modifier"` ou `"Supprimer"` au sein des gestionnaires `creerRadioBtnActionPerformed()`, `modifierRadioBtnActionPerformed()` ou `supprimerRadioBtnActionPerformed()`.

Copyright © 2016 Eyrolles.

f. Modifiez le code du gestionnaire `validerBtnActionPerformed()` en y insérant les tests suivants :

```
String nom = nomAsaisir.getText();
String prenom = prenomAsaisir.getText();
if (choixAction.equals("Creer")) {
      creerUnEtudiant(nom, prenom);
}
else if (choixAction.equals("Modifier")) {
      modifierUnEtudiant(nom, prenom);
}
else if (choixAction.equals("Suprimer")) {
      supprimerUnEtudiant(nom, prenom);
}
```

g. Écrivez les fonctions `creerUnEtudiant()`, `modifierUnEtudiant()` et `supprimerUnEtudiant()` en vous inspirant des codes fournis au chapitre 10, « Collectionner un nombre indéterminé d'objets », sections « Les dictionnaires » et « Les fichiers d'objets ».

L'aide contextuelle

Exercice 12.5 Lorsque le curseur de la souris survole :

- `creerRadioBtn`, le message affiché dans la zone Info est « Créer un nouvel étudiant » ;
- `modifierRadioBtn`, le message affiché dans la zone Info est « Modifier un étudiant, vous devez connaître son nom et son prénom » ;
- `supprimerRadioBtn`, le message affiché dans la zone Info est « Supprimer un étudiant, vous devez connaître son nom et son prénom ».

Écrivez les gestionnaires d'événements associés qui réalisent ces différents affichages.

Exercice 12.6 Le survol de la `JComboBox choixPeriode` ne permet pas d'afficher correctement un message dans la zone Info. En effet, lorsque la liste des périodes est déroulée, le message affiché dans la zone Info peut être erroné. Par exemple, le survol de l'item Semestre 4 affiche le message « Info : Quitter l'application ». Ceci s'explique par le fait que la liste déroulante s'affiche au-dessus du bouton Quitter. Le survol de la souris est capturé par ce dernier. Pour corriger ce défaut d'affichage, la marche à suivre est la suivante :

a. Déclarez un drapeau `etatComboBox` comme propriété de la classe `CursusSwing` et initialisez-le à `false`. L'état `false` correspond à une `JComboBox` fermée, l'état `true` à une `JComboBox` déroulée.

b. Ajoutez le gestionnaire d'événements `PopupWillBecomeVisible` à la `JComboBox choixPeriode`. Dans ce gestionnaire, initialisez le drapeau `etatComboBox` à `true`. Affichez également le message « Info : Choisir la période de validation des notes » dans la zone Info.

Copyright © 2016 Eyrolles.

c. Ajoutez le gestionnaire d'événements `PopupWillBecomeInvisible` à la `JComboBox choixPeriode`. Dans ce gestionnaire, initialisez le drapeau `etatComboBox` à `false`. Affichez également le message « Info : Tous les champs doivent être renseignés ! » dans la zone Info.

d. Dans les gestionnaires de type `MouseEntered`, vérifiez l'état du drapeau `etatComboBox`. S'il est à `true`, affichez dans la zone Info le message « Info : Choisir la période de validation des notes » sinon, affichez le message relatif au composant concerné.

Une boîte Message

Exercice 12.7 Lorsqu'un étudiant est ajouté ou supprimé au fichier d'objets, lorsqu'il est modifié ou encore s'il n'est pas possible de le modifier, une alerte doit s'afficher à l'écran. Elle se présente sous la forme suivante (figure 12-45) :

Figure 12-45 *Un message est affiché pour indiquer le bon ou le mauvais déroulement du programme.*

a. Ajoutez au projet `GestionExempleExercice` une classe nommée `Message` basée sur une `JFrame`.

b. Dans le panneau Design, ajoutez trois `JLabel` et un `JButton` que vous nommerez respectivement `hautLabel`, `centreLabel`, `basLabel` et `okBtn`.

c. Le constructeur `Message()` prend en paramètre trois chaînes de caractères. Placez chacune d'entre elles au sein des composants `hautLabel`, `centreLabel` et `basLabel`

d. Écrivez le gestionnaire du bouton `okBtn` de façon à ce qu'il ferme la fenêtre sans quitter l'application.

e. Modifiez l'application `CursusSwing` pour y ajouter l'affichage d'une alerte lorsqu'un étudiant a été enregistré, modifié, supprimé ou s'il est inconnu.

Copyright © 2016 Eyrolles.

L'éditeur graphique version 2

L'objectif de cet exercice est d'améliorer et d'ajouter de nouvelles fonctionnalités à l'éditeur graphique présenté au cours de ce chapitre.

Dessiner un rectangle à la souris

Exercice 12.8 Reprennez l'algorithme de tracé d'un cercle décrit à la section « Dessiner un cercle à la souris » de ce chapitre.

a. À l'examen des quatre cas possibles de tracé d'un rectangle, écrivez dans la classe `FeuilleDeDessins` la fonction `dessinerUnRectangle()`.

b. Modifiez le gestionnaire `formMouseReleased` de manière à ce qu'un rectangle soit dessiné lorsque l'utilisateur sélectionne le bouton `rectangle`.

Effacer la dernière forme dessinée

Exercice 12.9 Pour effacer la dernière forme tracée, deux actions sont à réaliser :

- supprimer de la liste d'affichage la dernière forme ajoutée ;
- dessiner la liste d'affichage obtenue après suppression du dernier élément.

a. Dans la classe `ListeDeFormes`, insérez la méthode `supprimerLaDerniereForme()` qui calcule la longueur de la liste d'affichage et si cette dernière est non nulle, supprime le dernier élément de la liste.

Pour en savoir plus Les outils de gestion des `ArrayList` sont étudiés au chapitre 10, « Collectionner un nombre indéterminé d'objets », section « Les listes ».

b. Dans la classe `Main`, créez le gestionnaire d'événements associé au bouton `undo`. Placez-y les instructions qui suppriment la dernière forme tracée et qui affichent ensuite la liste sur la feuille de dessins en cours.

Ouvrir et enregistrer sous

Exercice 12.10 Lorsque l'utilisateur sélectionne l'item Enregistrer sous... du menu Fichier, une boîte de dialogue s'ouvre et l'invite à spécifier le répertoire et le nom du fichier d'enregistrement des données.

a. Insérez un composant `JMenuItem` au menu Fichier, nommez-le `itemSauverSous` dans le panneau Inspecteur et placez-y le texte « Enregistrer sous... ».

b. Ajoutez ensuite le gestionnaire d'événements `itemSauverSousActionPerformed()` de sorte que :

- un objet de type `JFileChooser` soit créé ;

Copyright © 2016 Eyrolles.

- une boîte de dialogue de type `showSaveDialog()` s'affiche et retourne le nom du fichier d'enregistrement dans une variable nommée `nomDuFichier` ;
- le fichier portant le nom choisi s'ouvre et les données s'enregistrent à l'intérieur.

Exercice 12.11 Lorsque l'utilisateur sélectionne l'item Ouvrir du menu Fichier, une boîte de dialogue s'ouvre et l'invite à rechercher dans le système de fichiers, le nom du fichier où sont enregistrées les données.

Pour réaliser ces différentes actions, vous devez modifier le gestionnaire d'événements `itemOuvrirActionPerformed()` développé au cours de ce chapitre de façon à :

- créer un objet de type `JfileChooser` ;
- afficher une boîte de dialogue de type `showOpenDialog()` ;
- récupérer le nom du fichier à ouvrir dans une variable nommée `nomDuFichier` ;
- ouvrir le fichier sélectionné et le lire pour en extraire les données ;
- stocker les données sous forme de liste d'affichage et dessiner les objets de la liste sur la feuille de dessins.

La fenêtre À propos

Exercice 12.12 La fenêtre À propos (voir figure 12-46) s'affiche lorsque l'utilisateur sélectionne l'item À propos du menu Aide.

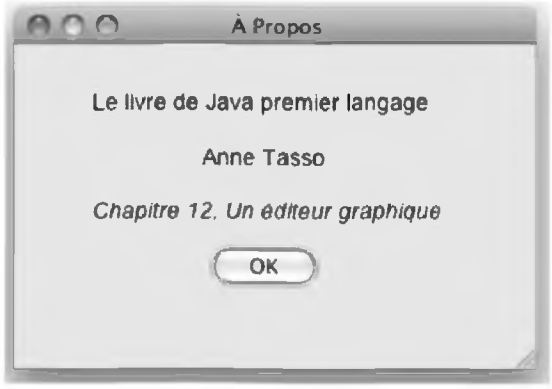


Figure 12-46 *La fenêtre À propos*

a. Pour afficher la fenêtre À propos, reprennez la classe `Message` développée au cours de l'exercice 12-7.

Pour en savoir plus Pour copier une classe d'un projet à un autre, reportez-vous à la section « Utilisation des outils de développement » de l'annexe « Guide d'installations ».

Copyright © 2016 Eyrolles.

b. Revenez à la classe `Main` pour y insérer le composant `JMenuItem` au sein de l'item Aide. Nommez-le `itemApropos` dans le panneau Inspecteur et placez-y le texte « À propos ».

c. Ajoutez ensuite le gestionnaire d'événements `itemAproposActionPerformed()` de façon à ce qu'il affiche la fenêtre « À propos » avec les messages tels que ceux qui sont affichés sur la figure 12-46.

Le projet : Gestion de comptes bancaires

L'objectif est de transformer l'application de gestion de comptes bancaires en mode commande, en une application basée sur une interface de communication conviviale. Plusieurs étapes sont nécessaires à cette transformation.

La gestion des entrées (saisie du numéro de compte, du type, etc.) se fait par l'intermédiaire de formulaires construits à l'aide du panneau Design de NetBeans. L'affichage des données d'un compte est également réalisé dans une fenêtre construite en partie par l'application.

Pour réaliser toutes ces transformations, il convient donc de modifier une grande partie l'application. Pour rester compréhensible et faire en sorte que le projet soit réalisable en un temps relativement raisonnable, nous avons choisi de vous fournir une partie des formulaires et du code.

Extension Web Vous trouverez tous les fichiers nécessaires à la réalisation de cette application dans le répertoire `Source/Projet/Chapitre12/SupportPourRealiserLeProjet` sur l'extension Web de cet ouvrage.

Cahier des charges

L'interface entre l'utilisateur et l'application de gestion de comptes bancaires utilise trois formulaires de saisie, représentés à la figure 12-47.

Le premier panneau présente les trois actions de gestion d'un compte bancaire : la création, la mise à jour et l'édition d'un compte dont le numéro est saisi par l'utilisateur (voir figure 12-47).

Lorsque l'utilisateur choisit de créer un compte, la fenêtre présentée à la figure 12-48 apparaît après validation du numéro de compte.

Pour créer un compte, il suffit de sélectionner son type en cochant la case Compte Courant ou Compte Entreprise et d'indiquer le montant déposé à l'ouverture du compte. La fenêtre se ferme automatiquement après validation.

Copyright © 2016 Eyrolles.

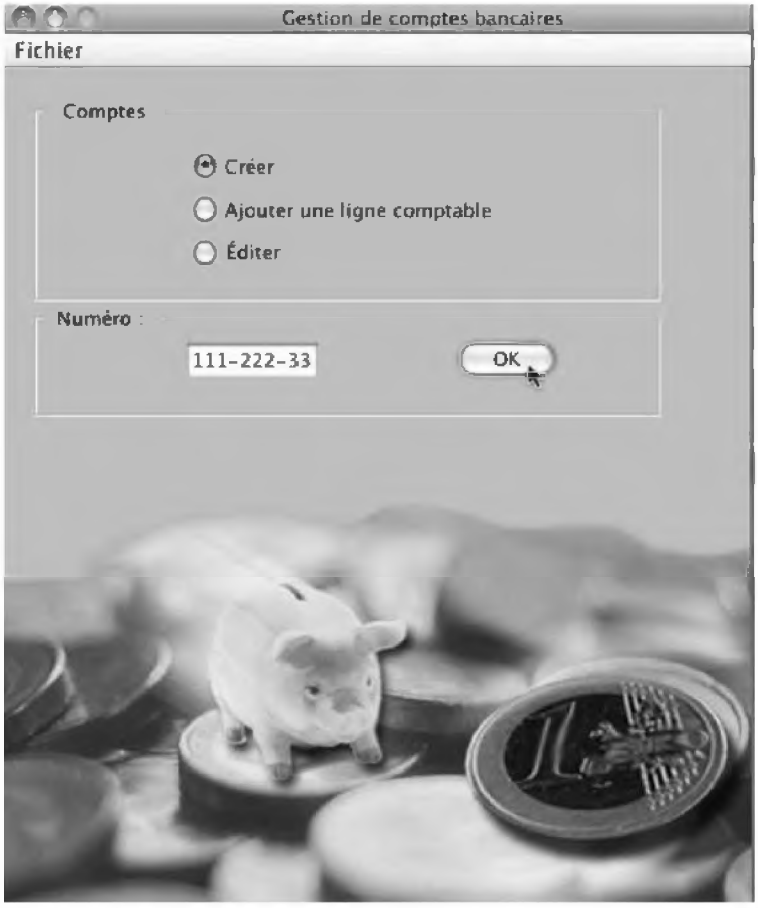


Figure 12-47 *La fenêtre d'accueil de l'application*

Figure 12-48
Création d'un compte

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

L'utilisateur revient alors à la fenêtre initiale où il peut choisir de créer des lignes comptables ou d'éditer le contenu d'un compte dont il a saisi le numéro (voir figure 12-47).

S'il choisit de créer des lignes comptables, la fenêtre représentée à la figure 12-49 apparaît.

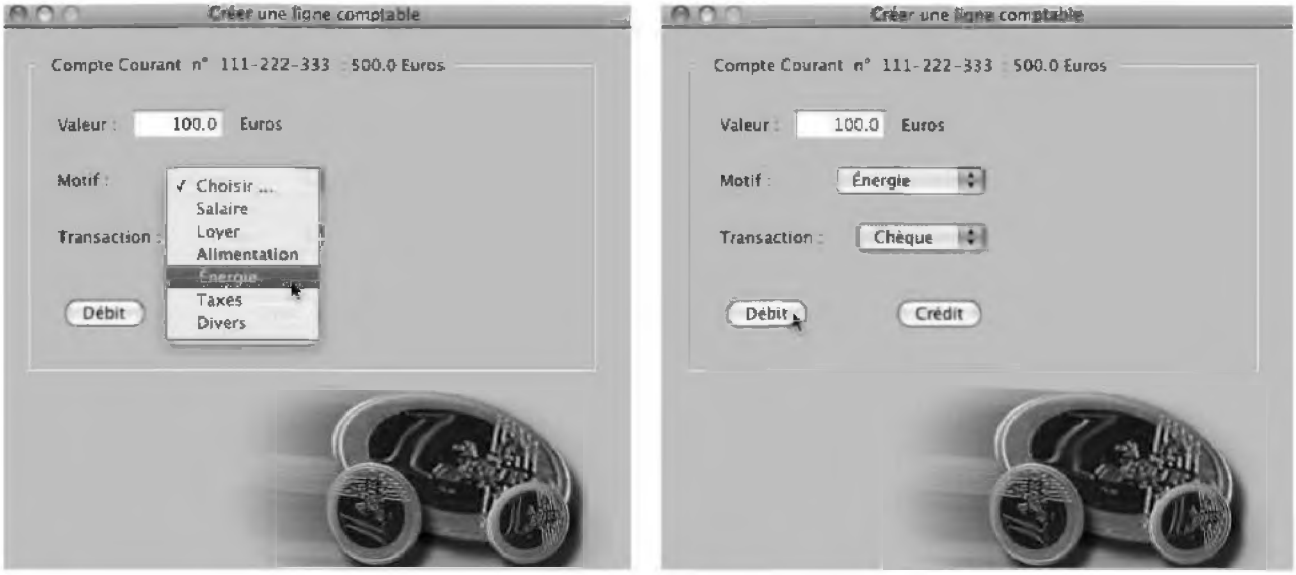


Figure 12-49 *Formulaire de saisie d'une ligne comptable*

Le formulaire de saisie d'une ligne comptable permet à l'utilisateur de fournir le montant de la transaction ainsi que son motif et son type. Pour ces deux derniers items, le choix est effectué à partir de listes déroulantes. La fenêtre se ferme automatiquement lorsque l'utilisateur clique sur l'un des deux boutons Débit ou Crédit.

L'utilisateur revient alors à la fenêtre initiale où il peut maintenant choisir d'éditer le compte dont il vient de saisir les données. Cette fenêtre n'est pas un formulaire de saisie, elle affiche simplement les données d'un compte bancaire (voir figure 12-50).

Structure de l'application

L'application se décompose en plusieurs classes Java. On distingue deux types de classes : celles reliées aux données traitées par l'application et celles permettant la création des formulaires.

Classes de traitement des données

Les classes `Compte`, `LigneComptable` et `FichierCompte` sont les trois classes de base qui vont nous permettre de traiter les données associées à un compte bancaire. Ces classes ont été, pour une grande part, écrites au cours des chapitres précédents.

Copyright © 2016 Eyrolles.

Figure 12-50 *Fenêtre d'édition d'un compte bancaire*

Pour simplifier la mise en œuvre de l'application, nous avons choisi de manipuler les comptes et leurs données associées en enregistrant chaque compte créé dans un fichier objet.

Ainsi, lorsque l'utilisateur crée un compte, un fichier objet portant le numéro du compte suivi de l'extension `.dat` est automatiquement créé après validation du formulaire Créer un compte.

Lorsque l'utilisateur souhaite ajouter une ligne comptable ou éditer le contenu d'un compte, l'application ouvre le fichier associé et stocke en mémoire le contenu dans un objet de type `Compte`. Si l'utilisateur ajoute des lignes comptables, celles-ci sont ensuite enregistrées dans le fichier associé, lorsque l'utilisateur clique sur les boutons Débit ou Crédit.

Les lignes comptables sont, quant à elles, traitées comme des listes de type `ArrayList`.

Vous trouverez les trois classes `Compte`, `LigneComptable` et `FichierCompte` dans le répertoire `Source/Projet/Chapitre12/SupportPourRealiserLeProjet` sur l'extension Web de l'ouvrage. Celles-ci ont été légèrement modifiées par rapport aux chapitres précédents. La lecture des commentaires vous permettra de mieux comprendre leur fonctionnement.

Copyright © 2016 Eyrolles.

Classes de création de formulaires

Les classes `Main`, `CompteDialogue`, `CompteEdit`, `LigneDialogue` et `LigneEdit` sont les cinq classes qui vont vous permettre de construire les formulaires et leurs interactions.

Pour construire graphiquement chacune de ces classes, reportez-vous aux différentes fenêtres représentées par les figures 12-47 à 12-50 et aux panneaux Inspecteur associés à chacune des cinq classes, présentées ci-après.

Mise en place des éléments graphiques

La classe Main

La classe `Main` hérite des fonctionnalités d'une `JFrame`. Elle contient la fonction `main()` et propose trois boutons radio (`creerRdBtn`, `modifierRdBtn` et `editerRdBtn`) qui permettent à l'utilisateur de choisir entre créer, ajouter une ligne comptable et éditer un compte. Ces trois boutons sont placés au sein du groupe de boutons nommé `compteGrp` (figure 12-51).

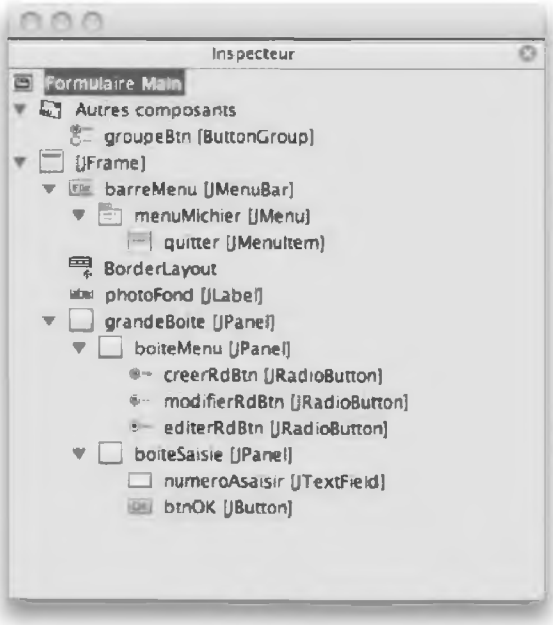


Figure 12-51 *Structure de la classe Main*

Le numéro du compte est fourni à l'application par l'intermédiaire d'un champ de saisie nommé `numeroAsaisir`. Le bouton `btnOK` est utilisé pour valider la saisie du numéro du compte bancaire et afficher l'une des trois fenêtres décrites ci-après.

Pour finir, le panneau associé à la classe `Main` contient une barre de menus contenant un entête `menuFichier`, composé d'un seul item nommé `quitter`.

Copyright © 2016 Eyrolles.

La classe CompteDialogue

La classe `CompteDialogue` hérite des fonctionnalités d'une `JFrame`. Elle ne contient pas de fonction `main()` et s'affiche à l'aide de son constructeur qui est appelé par la classe `Main` (figure 12-52).

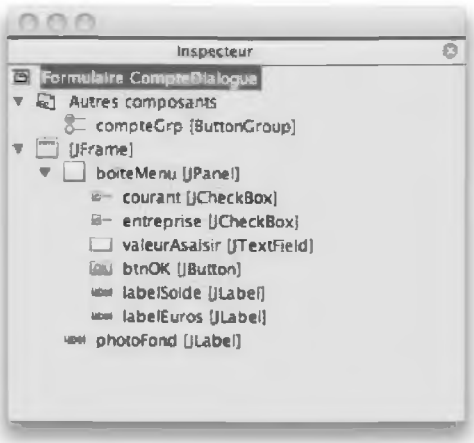


Figure 12-52 *Structure de la classe CompteDialogue*

La classe `CompteDialogue` est utilisée pour saisir les données nécessaires à la création d'un compte bancaire. Le choix du type du compte est réalisé grâce aux deux cases à cocher `courant` et `entreprise` placées au sein d'un groupe de boutons nommé `compteGrp`. La valeur du montant déposé à l'ouverture du compte est fournie par le champ de saisie `valeur Asaisir`.

La classe LigneDialogue

La classe `LigneDialogue` hérite des fonctionnalités d'une `JFrame`. Elle ne contient pas de fonction `main()` et s'affiche à l'aide de son constructeur qui est appelé par la classe `Main` (figure 12-53).

La classe `LigneDialogue` est utilisée pour saisir les données nécessaires à la création d'une ligne comptable.

La valeur de la transaction est fournie par le champ de saisie `valeurAsaisir`. Le choix du motif et du mode de l'opération comptable est réalisé grâce aux deux listes déroulantes `choixMotif` et `choixTransaction`.

Les boutons `creditBtn` et `debittBtn` sont utilisés pour valider les valeurs saisies. Si l'utilisateur clique sur `creditBtn`, la valeur saisie par l'intermédiaire du champ `valeur Asaisir` reste positive. S'il clique sur `debittBtn`, la valeur devient négative.

Copyright © 2016 Eyrolles.

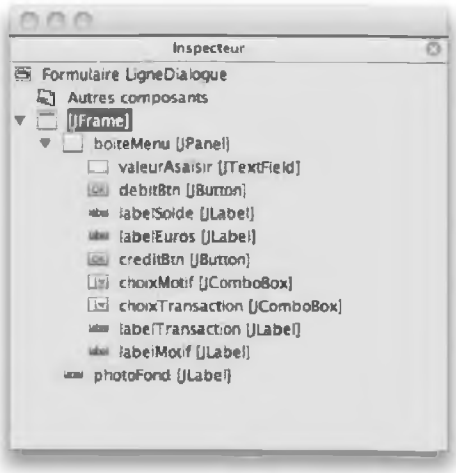


Figure 12-53 *Structure de la classe LigneDialogue*

La classe CompteEdit

La classe `CompteEdit` hérite des fonctionnalités d'une `JFrame`. Elle ne contient pas de fonction `main()` et s'affiche à l'aide de son constructeur qui est appelé par la classe `Main` (figure 12-54).

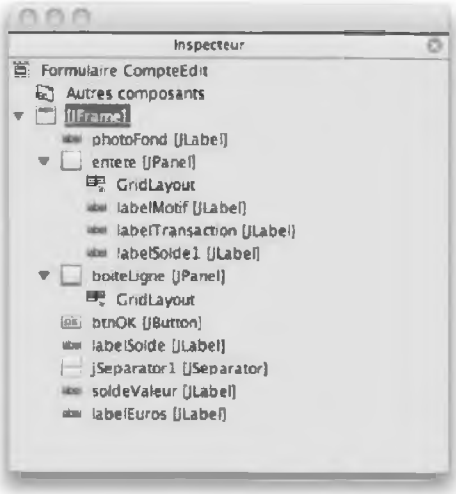


Figure 12-54 *Structure de la classe CompteEdit*

La classe `CompteEdit` est utilisée pour afficher toutes les informations relatives à un compte. Elle contient deux `Jpanel` : `entete` et `boiteLigne`. La boîte `entete` est le chapeau de la

Copyright © 2016 Eyrolles.

fenêtre d'édition du compte. Il contient pour chaque colonne, le nom des rubriques (Motif, Transaction et Valeur). La boîte `boiteLigne` est vide, mais elle se remplira par programme, d'autant de lignes comptables (`LigneEdit`) qu'il y a de lignes comptables enregistrées dans le fichier associé au compte bancaire.

Au-dessous des lignes comptables se trouve le label nommé `labelSolde` qui contient le solde du compte en cours d'édition.

La classe LineEdit

La classe `LigneEdit` hérite des fonctionnalités d'un `JPanel`. Son contenu ne s'affiche pas dans une fenêtre mais dans un panneau rectangulaire représentant le fond d'une ligne comptable (figure 12-55).

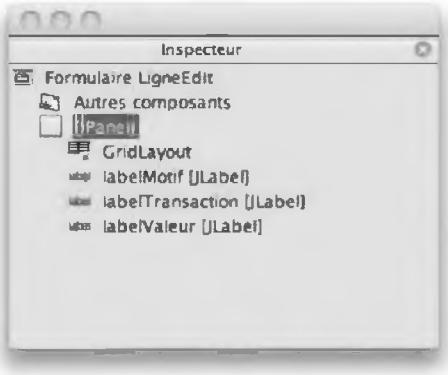


Figure 12-55 *Structure de la classe LigneEdit*

Les objets de type `LineEdit` sont créés et affichés au sein de la classe `CompteEdit` au moment de l'affichage des lignes comptables.

Définition des comportements

Chaque panneau contient ses propres comportements qu'il convient de décrire au sein de chacune des classes `Main`, `CompteDialogue`, `CompteEdit`, `LigneDialogue` et `LigneEdit`.

La classe Main

La classe `Main` définit deux propriétés : `choixAction` et `numeroCompte`.

- La propriété `choixAction` est initialisée dans les gestionnaires d'événements associés au trois boutons radio `creerRdBtn`, `modifierRdBtn` et `editerRdBtn`. Les valeurs d'initialisation sont respectivement `"Creer"`, `"Modifier"` ou `"Editer"`.

Copyright © 2016 Eyrolles.

- Le numéro du compte est récupéré par l'intermédiaire du champ de saisie `numeroAsaisir`, dans le gestionnaire d'événements associé au bouton `btnOK`. Dans ce même gestionnaire, la variable `choixAction` est testée et selon sa valeur les différentes actions sont menées.
  - Si `choixAction` est égale à `"Creer"`, la boîte de dialogue `CompteDialogue` est affichée par simple appel à son constructeur, avec le numéro du compte passé en paramètre.
  - Si `choixAction` est égale à `"Modifier"`, le fichier associé au compte est ouvert en lecture. Un objet de type `Compte` est créé et initialisé aux données enregistrées dans le fichier. La boîte de dialogue `LigneDialogue` est affichée par simple appel à son constructeur, avec le compte passé en paramètre.
  - Si `choixAction` est égale à `"Editer"`, le fichier associé au compte est ouvert en lecture. Un objet de type `Compte` est créé et initialisé aux données enregistrées dans le fichier. La fenêtre d'édition `CompteEdit` est affichée par simple appel à son constructeur, avec le compte passé en paramètre.

Pour finir, l'action de quitter l'application est insérée dans le gestionnaire de l'item `quitter`.

La gestion des erreurs

Pour éviter des erreurs de lecture de fichiers, vous pouvez, lors de l'ouverture des fichiers associés aux comptes, vérifier que celle-ci s'est bien déroulée. Dans le cas contraire, créez un message d'alerte en utilisant la classe `Message` développée à l'exercice 12-7 de ce chapitre.

La classe CompteDialogue

Le constructeur de la classe `CompteDialogue` affiche la `boiteMenu` avec un style `BorderTitre`. Le titre contient le numéro du compte. Le style d'encadrement est réalisé par les instructions suivantes :

```
Border cadre = BorderFactory.createTitledBorder("   Compte n' : " +
                                                 cpt + "    " );
boiteMenu.setBorder(cadre);
```

où la variable `cpt` correspond au numéro du compte passé en paramètre du constructeur `CompteDialogue()`.

La classe `CompteDialogue` définit deux propriétés : `typeCompte` et `numeroCompte`.

- La propriété `typeCompte` est initialisée dans les gestionnaires d'événements associés aux cases à cocher `courant` et `entreprise` aux valeurs `"Courant"` ou `"Entreprise"`, respectivement.
- La propriété `numeroCompte` est initialisée dans le constructeur, à la valeur passée en paramètre de ce dernier.

Copyright © 2016 Eyrolles.

Le gestionnaire d'événements du bouton `btnOk` réalise les actions suivantes :

- Récupérer la valeur du montant déposé à l'ouverture du compte par l'intermédiaire du champ de saisie `valeurAsaisir`. Stocker cette valeur dans une variable nommée `valeur Initiale`.
- Créer un objet de type `Compte` à l'aide de son constructeur. Les trois données `type Compte`, `numeroCompte` et `valeurInitiale` sont fournies en paramètres de ce dernier.
- Enregistrer le compte dans un fichier objet dont le nom porte le numéro du compte suivi de l'extension `.dat`.
- Fermer la fenêtre en cours sans quitter l'application.

La classe LigneDialogue

Le constructeur de la classe `LigneDialogue` affiche la `boiteMenu` avec un style `Border Titre`. Le titre contient le type du compte ainsi que le solde en cours. Toutes ces valeurs sont récupérées par l'intermédiaire du compte passé en paramètre du constructeur. Les instructions réalisant l'affichage du cadre sont similaires à celles présentées à la section précédente.

La classe `CompteDialogue` définit trois propriétés : `motif`, `transaction` et `compte`.

- Les propriétés `motif` et `transaction` sont initialisées dans les gestionnaires d'événements associés aux listes déroulantes `choixMotif` et `choixTransaction` aux valeurs sélectionnées par l'utilisateur.
- La propriété `compte` est initialisée dans le constructeur, à la valeur passée en paramètre de ce dernier.

Les gestionnaires d'événements des boutons `debitBtn` et `creditBtn` sont quasi identiques. Ils réalisent les actions suivantes :

- Récupérer la valeur du montant de la transaction par l'intermédiaire du champ de saisie `valeurAsaisir`. Stocker cette valeur dans une variable nommée `valeur`.
- Dans le gestionnaire du bouton `debitBtn`, rendre cette valeur négative.
- Créer un objet de type `LigneComptable` à l'aide de son constructeur. Les trois données `valeur`, `motif` et `transaction` sont fournies en paramètres de ce dernier.
- Modifier la propriété `ligne` du `compte` en cours de traitement en utilisant la méthode d'accès en écriture `setLigne()` définie au sein de la classe `Compte`.
- Enregistrer le compte dans un fichier objet dont le nom porte le numéro du compte suivi de l'extension `.dat`.
- Fermer la fenêtre en cours sans quitter l'application.

Copyright © 2016 Eyrolles.

La classe CompteEdit

La classe `CompteEdit` ne fait qu'afficher le contenu du compte passé en paramètre du constructeur.

La seule interaction est celle définie par le gestionnaire du bouton `btnOK` qui a pour action de fermer la fenêtre en cours sans quitter l'application.

Le constructeur de la classe `CompteEdit` affiche :

- La boîte `entete` avec un style `BorderTitre`. Le titre contient le type du compte ainsi que le solde en cours. Toutes ces valeurs sont récupérées par l'intermédiaire du compte passé en paramètre du constructeur. Les instructions réalisant l'affichage du cadre sont similaires à celles présentées à la section précédente.
- Toutes les lignes comptables stockées dans le compte passé en paramètre. Le contenu des lignes comptables est récupéré en utilisant la méthode d'accès en lecture `getLigne()` définie au sein de la classe `Compte`. Pour chaque ligne récupérée, un objet de type `LigneEdit` est créé et ajouté au conteneur `boiteLigne`. L'objet est créé en passant en paramètre la ligne comptable en cours de traitement.
- Le solde du compte est récupéré en utilisant la méthode d'accès en lecture `getSolde()` définie au sein de la classe `Compte`. Si le solde est négatif, sa valeur est affichée en rouge en bas de la fenêtre d'édition du compte.

La classe LigneEdit

La classe `LigneEdit` ne fait qu'afficher le contenu de la ligne comptable passée en paramètre du constructeur.

Le constructeur de la classe `CompteEdit` affiche les données en modifiant les trois labels `labelMotif`, `labelTransaction` et `labelValeur`. Les valeurs affichées sont récupérées en utilisant les méthodes d'accès en lecture `getMotif()`, `getMode()` et `getValeur()` définies au sein de la classe `LigneComptable`.

Le fond des fenêtres

Toutes les fenêtres de l'application présentent une image d'arrière-plan qui les caractérise. Ces images sont enregistrées dans un répertoire nommé `Ressources`. Elles ont pour nom `FondCompte.png`, `FondEdit.png`, `FondLigne.png` et `FondMain.png`.

L'affichage des images est réalisé comme suit :

```
ImageIcon iconPhoto = new ImageIcon("Ressources/nomImage.png");
photoFond.setIcon(iconPhoto);
```

Ces instructions sont placées au sein des constructeurs de chacune des classes affichant une fenêtre. L'objet `photoFond` est un composant de type `JLabel`, placé en bas de chacune des fenêtres.

Copyright © 2016 Eyrolles.

Copyright © 2016 Eyrolles.

Chapitre 13

Développer une application Android

La création d'applications pour mobiles et tablettes tactiles est aujourd'hui incontournable sur le marché du développement informatique. Ces appareils sont en effet dotés de systèmes d'exploitation performants qui permettent le développement d'interfaces utilisateur conviviales.

Le système d'exploitation mobile Android, en plein essor, utilise la technologie Java pour le développement de ses applications. Il existe également des environnements de développement adaptés qui simplifient leur création.

L'objectif de ce chapitre est de vous initier à la programmation d'applications Android et non de faire de vous un « expert », car ce seul manuel ne suffirait pas.

Pour cela, nous vous proposons à la section « Comment développer une application mobile ? » de développer, avec l'interface de programmation Android Studio, une première application Android très simple. Nous étudierons également la structure de base d'une telle application.

Dans la section « L'application Liste de courses », nous présenterons pas à pas comment construire une application conviviale qui propose à l'utilisateur d'éditer sur son mobile sa propre liste des courses.

Pour finir, nous présenterons à la section « Publier une application Android » les différentes étapes qui vous permettront de déposer votre application sur un serveur dédié aux applications pour mobile Android.

Comment développer une application mobile ?

Une application mobile est un logiciel, un programme, que vous devez télécharger depuis Internet, puis installer sur votre mobile. Le téléchargement s'effectue soit à partir de votre mobile, soit depuis votre ordinateur.

Copyright © 2016 Eyrolles.

Remarque Dans ce chapitre, les termes « applications mobiles » recouvrent par extension les applications pour téléphones mobiles et tablettes tactiles. Le mode de programmation de ces deux appareils est quasi identique, seul le système d'exploitation importe dans le choix du langage de programmation.

Les mobiles basés sur un système d'exploitation de type Android (HTC, Samsung, Sony) utilisent des langages de programmation tels que Java ou AS3-Air, alors que ceux basés sur un système d'exploitation de type iOS (iPhone, iPad) utilisent des langages de programmation tels que Objective C ou également AS3-Air.

Ainsi, le développement d'une application Android requiert l'utilisation d'un environnement de développement spécifique. Dans cet ouvrage, nous avons choisi de présenter l'IDE Android Studio proposé par Google. Pour installer cet environnement, reportez-vous à l'annexe « Guide d'installations », section « Développer des applications Android avec Android Studio ».

Il existe aussi d'autres plug-ins Android notamment pour Eclipse, IntelliJ ou encore l'environnement de développement AIDE (pour *Android Java IDE*) qui a la particularité de s'installer directement sur une tablette Android.

Bonjour le monde : votre première application mobile

L'objectif est de créer une toute première application Android afin de se familiariser avec les outils de développement, de compilation, d'exécution et comprendre la structure générale d'une telle application.

Extension Web Vous trouverez tous les codes de cette première application dans le répertoire `Sources/Exemple/Chapitre13/AndroidStudio/BonjourLeMonde`.

Création d'un projet Android

La création et l'exécution d'une application Android s'effectuent dans le cadre d'un projet Android Studio au sein duquel sont regroupées toutes les ressources nécessaires à la bonne marche de l'application.

Pour créer un projet Android, une fois l'IDE Android Studio installée et lancée, vous devez cliquer sur la rubrique « Start a new Android Studio project » (figure 13-1) sur le panneau d'accueil ou sélectionner l'item Nouveau projet du menu Fichier, si d'autres projets sont déjà ouverts.

Copyright © 2016 Eyrolles.

Figure 13-1 *Le panneau d'accueil de l'application Android Studio*

Dans la boîte de dialogue Create New Project qui apparaît (figure 13-2), vous devez :

1. Saisir le nom de l'application (ici, `BonjourLeMonde`).
2. Définir le nom du package (ici, `android.PremierProjet`).
3. Si le dossier d'enregistrement du projet indiqué par défaut ne vous convient pas, spécifier un autre emplacement en cliquant sur le bouton « ... » de la rubrique **Project location**.
4. Valider l'enregistrement du projet en cliquant sur le bouton Next.

Pour en savoir plus Toutes les informations nécessaires à l'installation de Android Studio sur votre machine (Windows, Mac OS ou Linux) sont fournies dans l'annexe « Contenu et exploitation du CD-Rom », section « Installation d'un environnement de développement ».

Vous devez ensuite choisir (figure 13-3) la version de l'interface de programmation (API, *Application Programming Interface*) sur laquelle vous souhaitez exécuter la simulation – ici, API 8.0 Android 2.2 (Froyo). Le choix de la version de l'API se fait en fonction de la cible que vous souhaitez atteindre. Plus la version est récente, plus le nombre de « clients » potentiels diminue. Cliquez ensuite sur Next en ne cochant pas les autres options (TV, Wear...).

Copyright © 2016 Eyrolles.

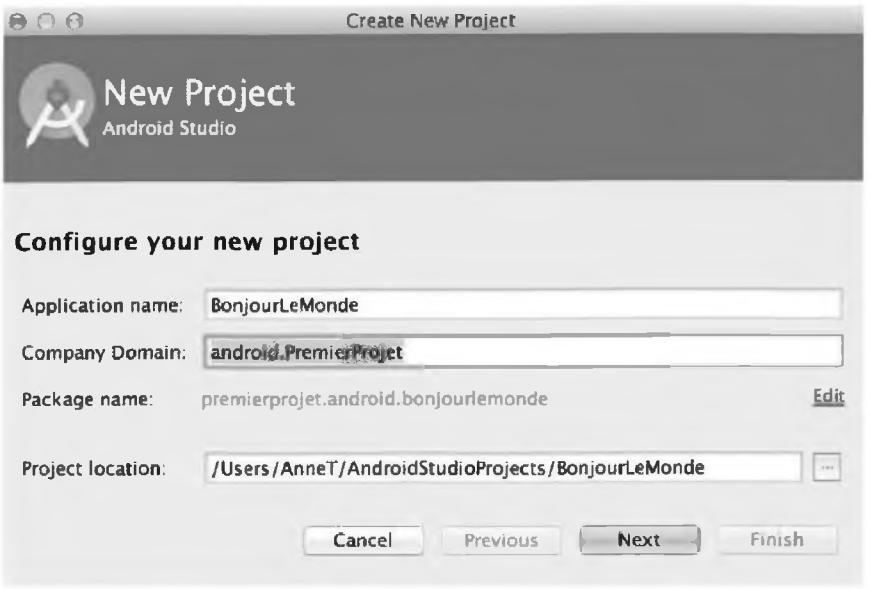


Figure 13-2 *Le panneau Create New Project, étape New Project*

Figure 13-3 *Le panneau Create New Project, étape Target Android Devices*

Copyright © 2016 Eyrolles.

Le panneau suivant (figure 13-4) vous demande de sélectionner un modèle de présentation pour votre application (Template). Sélectionnez Empty Activity puis cliquez sur Next.

Figure 13-4 *Le panneau Create New Project, étape Add an activity to Mobile*

Le dernier écran de personnalisation de votre application (figure 13-5) apparaît, modifiez les noms de l'activité (ici, Activity Name : Main) et celui des ressources (ici, Layout Name : main), puis cliquez sur Finish.

La génération du projet Android entraîne la création d'une application basique « Hello World » qui contient des fichiers par défaut, organisés selon une arborescence assez complexe. Examinons plus précisément le contenu de cette arborescence (figure 13-6).

Copyright © 2016 Eyrolles.

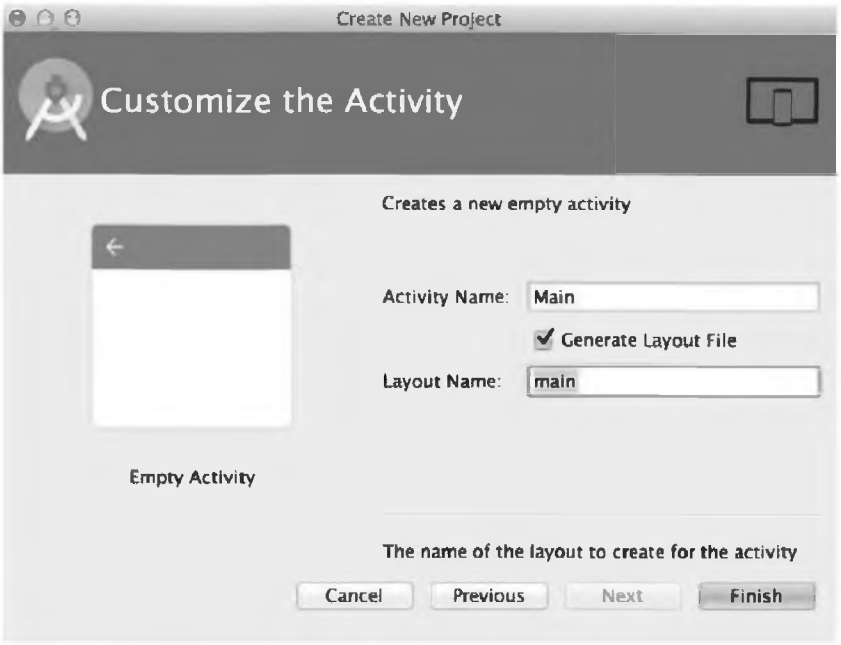


Figure 13-5 *Le panneau Create New Project, étape Customize the Activity*

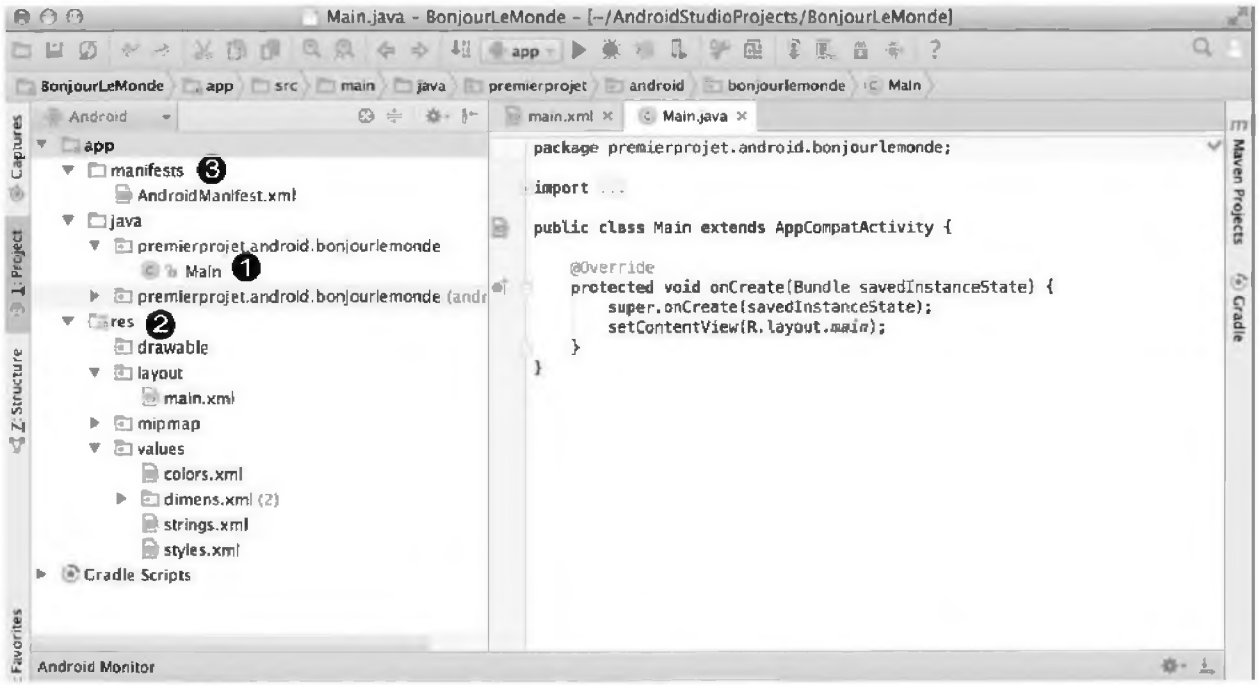


Figure 13-6 *Le panneau Create New Project, étape Customize the Activity*

Copyright © 2016 Eyrolles.

Arborescence du projet Android

Quelle que soit l'interface de développement utilisée, un projet de type Android possède une structure bien définie. Grâce à cette hiérarchie, les ressources et les codes sources sont facilement repérables, et le projet devient plus lisible puisque bien organisé.

L'arborescence automatiquement générée par Android Studio se présente sous la forme suivante (figure 13-6, repères ❶ à ❸).

❶ Le répertoire `Java` contient toutes les classes Java nécessaires au bon fonctionnement de l'application. La classe principale, créée en même temps que le projet, est visible en double-cliquant sur la ressource `Main` située dans l'arborescence du projet. Nous examinerons ce fichier plus en détail à la section « Les classes Java ».

❷ Le répertoire `res` contient des sous-répertoires dans lesquels sont enregistrées toutes les ressources utiles à l'application comme des images (*drawable*) ou des fichiers descriptifs des composants d'affichage utilisés par l'application (*layout*). Il contient également des valeurs textuelles (*values*). Ces éléments seront détaillés à la section « Les fichiers descriptifs ».

❸ Le répertoire `manifests` contient des fichiers importants dont `AndroidManifest.xml` dans lequel sont définis les activités et services proposés par l'application. Nous examinerons son utilité dans la section « L'application Liste de courses » à la fin de ce chapitre.

Compilation et exécution de projet

Une fois le projet compilé en cliquant sur le triangle vert proposé par l'interface, Android Studio lance l'exécution de l'application en ouvrant l'émulateur Android défini lors de la création du projet (voir section « Développer des applications Android avec Android Studio – Créer un émulateur Android », dans l'annexe Guide d'installations en fin d'ouvrage).

Une fois ouvert, le téléphone affiche l'application `BonjourLeMonde` (figure 13-7).

Figure 13-7
L'application BonjourLeMonde s'affiche après chargement de l'émulateur.

Copyright © 2016 Eyrolles.

Remarque Il n'est pas nécessaire de fermer le simulateur Android à chaque nouvelle exécution, car il met du temps à se lancer. Il charge l'application modifiée par vos soins, à chaque fois que vous lancez une compilation.

L'application s'intitule `BonjourLeMonde` et affiche le texte `Hello World!` sur un écran blanc. Ces valeurs ont été créées par défaut lors de la construction du projet. Elles sont modifiables grâce aux fichiers ressources, que nous allons à présent détailler.

Les fichiers ressources

Les fichiers ressources sont utilisés pour stocker les différents médias employés par l'application. Il peut s'agir de photos, de sons ou de vidéos. Mieux encore, il est possible de définir la façon dont vous souhaitez agencer les composants graphiques de votre application en décrivant leur organisation à l'aide d'une structure XML.

XML en quelques mots

L'utilisation d'un fichier au format XML permet de simplifier la façon d'agencer les composants au sein d'une application Android.

En effet, le format XML (*eXtensible Markup Language*) est un langage de description de données. Grâce à sa structure, il permet d'organiser les données en les nommant et en les ordonnant selon une hiérarchie qui décrit l'agencement des composants au sein de l'application.

La syntaxe du langage XML est assez proche de celle du langage HTML, composée de balises et d'attributs dont le nom reflète le composant graphique utilisé par l'application. Ainsi, les lignes suivantes :

```
<Button
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="OK"

/>
```

ont pour résultat de créer un bouton, grâce à la balise prédéfinie `<Button>`, qui aura une forme particulière décrite par les attributs `android:layout_width` et `android:layout_height`. La valeur `fill_parent` indique au gestionnaire d'affichage que le bouton a la même taille que son parent soit, le plus souvent, celle de l'écran du téléphone. La valeur `wrap_content` indique, quant à elle, que le bouton s'adapte à son contenu.

Ici, l'attribut `layout_width` recevant la valeur `fill_parent`, la largeur du bouton correspond à celle de l'écran du téléphone (figure 13-8, repère ❶). L'attribut `layout_height` ayant pour valeur `wrap_content`, la hauteur du bouton s'adapte à son contenu, c'est-à-dire à la hauteur du texte `OK`, puisque la balise `android:text` prend la valeur `OK`.

Copyright © 2016 Eyrolles.

Remarque Il n'est pas nécessaire de fermer le simulateur Android à chaque nouvelle exécution, car il met du temps à se lancer. Il charge l'application modifiée par vos soins, à chaque fois que vous lancez une compilation.

L'application s'intitule `BonjourLeMonde` et affiche le texte `Hello World!` sur un écran blanc. Ces valeurs ont été créées par défaut lors de la construction du projet. Elles sont modifiables grâce aux fichiers ressources, que nous allons à présent détailler.

Les fichiers ressources

Les fichiers ressources sont utilisés pour stocker les différents médias employés par l'application. Il peut s'agir de photos, de sons ou de vidéos. Mieux encore, il est possible de définir la façon dont vous souhaitez agencer les composants graphiques de votre application en décrivant leur organisation à l'aide d'une structure XML.

XML en quelques mots

L'utilisation d'un fichier au format XML permet de simplifier la façon d'agencer les composants au sein d'une application Android.

En effet, le format XML (*eXtensible Markup Language*) est un langage de description de données. Grâce à sa structure, il permet d'organiser les données en les nommant et en les ordonnant selon une hiérarchie qui décrit l'agencement des composants au sein de l'application.

La syntaxe du langage XML est assez proche de celle du langage HTML, composée de balises et d'attributs dont le nom reflète le composant graphique utilisé par l'application. Ainsi, les lignes suivantes :

```
<Button
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="OK"

/>
```

ont pour résultat de créer un bouton, grâce à la balise prédéfinie `<Button>`, qui aura une forme particulière décrite par les attributs `android:layout_width` et `android:layout_height`. La valeur `fill_parent` indique au gestionnaire d'affichage que le bouton a la même taille que son parent soit, le plus souvent, celle de l'écran du téléphone. La valeur `wrap_content` indique, quant à elle, que le bouton s'adapte à son contenu.

Ici, l'attribut `layout_width` recevant la valeur `fill_parent`, la largeur du bouton correspond à celle de l'écran du téléphone (figure 13-8, repère ❶). L'attribut `layout_height` ayant pour valeur `wrap_content`, la hauteur du bouton s'adapte à son contenu, c'est-à-dire à la hauteur du texte `OK`, puisque la balise `android:text` prend la valeur `OK`.

Copyright © 2016 Eyrolles.

Figure 13-8 *L'attribut layout_width est utilisé pour modifier la largeur d'un bouton*

En revanche, initialiser l'attribut `android:layout_width` à la valeur `wrap_content` comme suit :

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="OK"
/>
```

a pour conséquence de créer un bouton de largeur plus petite, qui sera ainsi adaptée à la taille de la chaîne de caractères `OK` (figure 13-8, repère ❷).

Les ressources layout

Les ressources relatives à l'agencement des composants graphiques utilisés par votre application sont stockées au sein de fichiers XML, dans le répertoire `layout`.

Remarque

Le nom d'un fichier ressource a pour extension `.xml`. Ce nom est ensuite utilisé par Android Studio pour générer des fichiers nécessaires à la bonne marche de l'application. Il ne doit comporter ni majuscule, ni espace, sous peine de créer des erreurs de syntaxe et donc de rendre impossible l'exécution de l'application.

Examinons plus attentivement le fichier `main.xml`, construit par défaut par Studio Android à la création du projet :

```
<?xml version="1.0" encoding="utf-8"?>

< RelativeLayout

      xmlns:android=http://schemas.android.com/apk/res/android
  android:paddingBottom="16dp"
      android:paddingLeft="16dp"
      android:paddingRight="16dp"
      android:paddingTop="16dp"
      android:layout_width="match_parent"
```

Copyright © 2016 Eyrolles.

```
        android:layout_height="match_parent"
>
    <TextView
        android:layout_width="wrap_parent"
        android:layout_height="wrap_content"
        android:text="Hello World, Main"
    />
</RelativeLayout>
```

Grâce aux informations stockées dans ce fichier, nous observons que l'application construite par défaut utilise un gestionnaire d'affichage de type `RelativeLayout` qui contient un composant de type `TextView`.

Un gestionnaire d'affichage (*layout* en anglais) peut être vu comme un conteneur de composants régi par des règles d'affichage qui lui sont spécifiques. Ainsi, lorsque nous utilisons un `RelativeLayout`, nous demandons à l'application d'afficher les composants qu'il contient, relativement aux autres, ou à leur parent.

La balise `RelativeLayout` est composée ici de cinq attributs :

1. `xmlns:android` (traduire `xmlns` par *XML name space* ou « espace de nom XML ») permet de certifier que tous les noms de balises et leurs attributs utilisés dans ce fichier sont régis par les spécifications Android ;
2. `android:paddingBottom` définit le remplissage à appliquer en bas du composant ;
3. `android:paddingTop` définit le remplissage à appliquer en haut du composant ;
4. `android:paddingLeft` définit le remplissage à appliquer à gauche du composant ;
5. `android:paddingRight` définit le remplissage à appliquer à droite du composant.

Remarque La valeur `16dp` attribuée par défaut aux balises `padding...` est définie dans le fichier `dimens.xml` du répertoire `values`.

`RelativeLayout` ne contient ici qu'un seul et unique composant. Il s'agit d'un composant `TextView`, utilisé pour afficher une zone de texte dont la largeur et la hauteur sont définies par les attributs `layout_width` et `layout_height`, respectivement. Le texte affiché est initialisé à la valeur transmise à l'attribut `android:text`, soit ici `Hello World!`. Vous pouvez dès à présent modifier le texte. Écrivez, par exemple, `Une toute première application Android` (voir figure 13-9) pour voir votre première application afficher un texte plus original.

Remarque Il existe d'autres types de layouts, comme `LinearLayout`, qui propose de placer les éléments qu'il contient les uns après les autres, dans l'ordre de définition des balises. Ses attributs sont, par exemple, `android:orientation`.

Copyright © 2016 Eyrolles.

Les ressources values

Le répertoire `values` contient des fichiers XML qui décrivent les variables utilisées par l'application. Les variables de type `String` sont définies dans un fichier nommé `strings.xml`, et les tableaux dans un fichier nommé `array.xml`.

Le fichier `strings.xml`, créé par Android Studio, contient les lignes suivantes :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">BonjourLeMonde</string>
</resources>
```

Il existe par défaut une chaîne de caractères nommée `app_name` qui contient le nom de l'application. Il s'agit du nom affiché (`BonjourLeMonde`) en titre de l'application. Pour modifier ce titre (voir figure 13-8), il suffit donc de changer la valeur de `app_name` comme suit :

```
<string name="app_name">BonjourLeMonde</string>
```

Figure 13-9 *Le texte et le titre de l'application sont modifiables grâce aux fichiers ressources*

Il est possible de déclarer d'autres chaînes de caractères dans le fichier `strings.xml`. Pour cela, il suffit d'écrire par exemple :

```
<resources>
    <string name="app_name">BonjourLeMonde</string>
    <string name="uneCouleur">rouge</string>
</resources>
```

La variable ressource `uneCouleur` ainsi définie est de type `String` et contient la chaîne de caractères `rouge`.

Les ressources drawable

Les ressources stockées dans le répertoire `drawable` correspondent à des photos et des graphiques utilisés par votre application. Pour insérer tous les médias nécessaires au bon fonctionnement de votre application dans le répertoire `drawable`, il vous suffit de sélectionner les

Copyright © 2016 Eyrolles.

fichiers images ou vidéo dans une fenêtre de votre système d'exploitation et de les glisser-déposer dans le répertoire `drawable` d'IDE (figure 13-10).

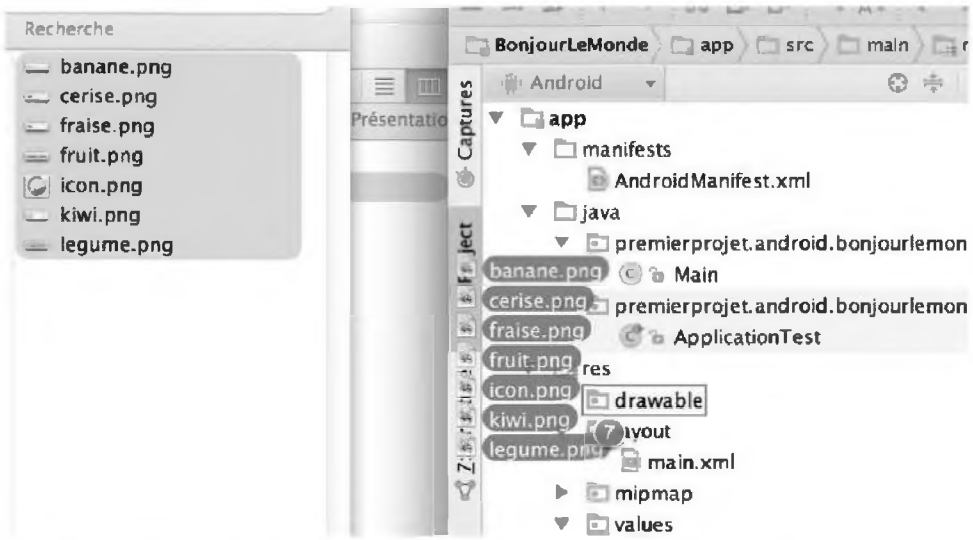


Figure 13-10 *Insérer des médias dans le répertoire drawable par glisser-déposer*

Les classes Java

Outre les fichiers ressources, un projet Android comprend des fichiers Java qui font que votre application va réagir aux actions de l'utilisateur. Ces fichiers contiennent les lignes de code Java qui indiquent à l'application quelle action réaliser lorsque l'utilisateur clique sur un composant ou le déplace.

Le fichier Main.java

La classe `Main.java` est créée en même temps que le projet et porte le nom de l'activité que vous avez spécifié à l'étape 4 de la création du projet (section précédente « Création d'un projet Android »).

La classe `Main` constitue le point d'entrée de votre application ; elle est composée des lignes suivantes :

```
package premierProjet.android.bonjourLeMonde;
import …;
// ❶ La classe Main hérite de la classe AppCompatActivity
public class Main extends AppCompatActivity {
  @Override
  // ❷ La méthode onCreate() est appelée à la création
  // de l'application
```

Copyright © 2016 Eyrolles.

```
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // ❸ Afficher le layout défini au sein du fichier main.xml
        setContentView(R.layout.main);
    }
}
```

❶ Après avoir importé toutes les bibliothèques utiles à la bonne marche de l'application, la classe `Main` est définie comme héritant des comportements de la classe `AppCompatActivity`. Cette dernière est une classe prédéfinie du langage Java pour Android ; elle comporte tous les outils nécessaires à la fabrication et à l'affichage de votre application mobile.

❷ Au lancement de l'application, la toute première méthode appelée est la méthode `onCreate()`. Elle est définie dans la classe `AppCompatActivity` et peut être vue « plus simplement » comme l'équivalent d'un constructeur. Elle comporte toutes les instructions d'initialisation demandées au lancement d'une application Android. Il n'est pas utile, dans cet ouvrage, d'entrer dans le détail de ces instructions. Les concepteurs Java ne le souhaitent pas non plus et nous autorisent à « redéfinir » la méthode `onCreate()` en plaçant, juste avant sa définition, le terme `override`.

Remarque Le terme `override` est utilisé à chaque fois que nous « redéfinissons » une méthode déjà définie au sein d'une classe héritée. Il ne s'agit pas d'une surcharge de méthode puisque les deux méthodes utilisées comportent chacune les mêmes paramètres en type et en nombre (voir chapitre 8, « Les principes de la programmation objet », section « Les méthodes d'accès aux données »).

Les téléphones mobiles Android sont multitâches. Il est ainsi possible de consulter ses courriels tout en écoutant de la musique. Chaque application (lire ses courriels, écouter de la musique) est une tâche qui doit être capable de gérer son fonctionnement, même si elle n'est pas affichée à l'écran. Le paramètre `Bundle saveInstanceState` est utilisé pour réaliser la sauvegarde de l'environnement d'exécution d'une application. Grâce à lui, l'application est capable de mémoriser et de retrouver son état lorsque l'utilisateur passe d'une application à une autre.

❸ Après exécution des instructions de la méthode `onCreate()` définie dans la classe héritée (`super.onCreate()`), la méthode `setContentView()` est enfin exécutée. Comme son nom l'indique, elle met à jour le contenu de la « vue » à afficher, c'est-à-dire celui de l'interface graphique de votre application. Ce contenu est accessible grâce à la valeur `R.layout.main` passée en paramètre.

Pour en savoir plus Les notions d'activités et de vues sont examinées plus attentivement à la section « Cycle de vie d'une application Android » ci-après.

Copyright © 2016 Eyrolles.

Le paramètre `R.layout.main` est construit par Android Studio à partir des répertoires et des fichiers du projet. Ainsi, les termes `layout` et `main` indiquent à la méthode `setContentView()` qu'il lui faut afficher les composants décrits dans le fichier `main.xml` stocké dans le répertoire `layout` du projet, soit ici le texte `Une toute première application android`.

Le fichier R.java

Une fois les fichiers ressources créés et lorsque l'application est « compilée-exécutée », Android Studio génère un fichier Java nommé `R.java` (R comme Ressources). Ce fichier, généré par l'IDE, est éditable mais non modifiable. Examinons son contenu :

```
public final class R {
    public static final class attr {
    }
    public static final class layout {
        public static final int main=0x7f020000;
    }
    public static final class string {
        public static final int app_name=0x7f030000;
    }
}
```

La classe `R.java` définit des classes internes `static` (`layout`, `string`) au sein desquelles sont créées des variables `static` nommées respectivement `main` et `app_name`.

Le nom des classes et des variables est à mettre en correspondance directe avec celui des répertoires et des fichiers qui se situent dans l'arborescence du projet :

- `main` car il existe un fichier `main.xml` dans le répertoire `layout` ;
- `app_name` parce que cette variable est définie dans le fichier `string.xml` du répertoire `values`.

Les données `main` et `app_main` sont créées par Android Studio afin d'établir une correspondance entre les ressources décrites dans l'arborescence du projet et le code Java de l'application. Ces données sont en réalité des constantes (`static final`). À chaque constante est attribuée une valeur hexadécimale calculée par Android Studio, chacune correspondant à un identifiant unique qu'il sera possible de récupérer au sein du code Java.

Pour accéder aux ressources, comme nous le verrons dans les deux exemples ci-après, il suffit d'utiliser dans le code Java les variables en indiquant leur chemin d'accès au package, soit : `R.layout.main` et `R.string.app_name`.

Cycle de vie d'une application Android

L'exécution d'une application Android passe par la création d'une activité qui contrôle son affichage par l'intermédiaire d'une vue. Au lancement de l'application, une première activité est créée. Tout au long de son exécution, l'activité va modifier son affichage en fonction des actions de l'utilisateur.

Copyright © 2016 Eyrolles.

Ce parcours d'exécution est appelé « cycle de vie ». Il se présente de la façon suivante (figure 13-11).

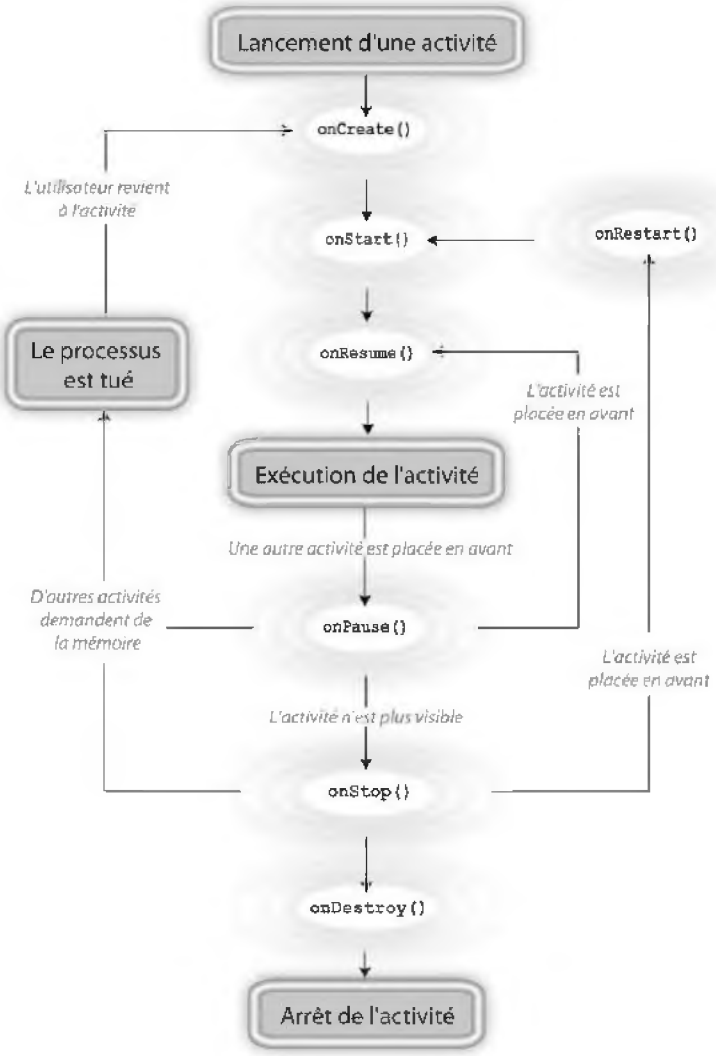


Figure 13-11 *Cycle de vie d'une application Android*

Copyright © 2016 Eyrolles.

Les activités (Activity) et les vues (View)

Une activité (`Activity`) est la première couche d'une application Android. C'est elle qui gère son cycle de vie et les événements qui lui sont associés. À la création d'une activité, ses composants graphiques sont affichés par l'intermédiaire de la méthode `onCreate()`.

Les vues (`View`) sont le support graphique d'une activité ; elles affichent les composants graphiques définis par l'intermédiaire du fichier `main.xml`. Toute activité possède donc des vues qui s'affichent selon les règles spécifiées par le gestionnaire d'affichage.

Lorsqu'une seconde activité est lancée en parallèle, elle possède à son tour des vues avec des règles d'affichage. Elle gère également son propre cycle de vie. Lors de son lancement, la première activité émet un événement `onPause`, puis `onStop`, alors que la seconde émet l'événement `onCreate` et devient visible en passant au premier plan.

Ainsi, par exemple, pour écrire un texto, vous lancez l'activité correspondant à l'écriture de SMS. Une fois ouverte, elle se place en haut de la liste des activités en cours, pour être visible. Les composants graphiques sont placés dans un système de vues, associé à l'activité `Texto`, selon un ordre prédéfini par le gestionnaire d'affichage, soit, par exemple, la liste des précédents messages envoyés et reçus par le contact, un champ de texte de saisie du texto en cours et un bouton Envoyer pour transmettre le message. Cette activité est au premier plan ; vous pouvez l'utiliser et agir directement sur ces composants.

Si vous écoutez de la musique en même temps, l'application qui gère les médias de votre téléphone est exécutée au travers d'une activité présente en mémoire, mais qui n'est pas visible. Elle est placée sous l'activité `Texto` dans la liste. Dans ce cas, l'activité est considérée comme mise en pause (`onPause`). Vu qu'elle est située en dessous d'une autre activité, il n'est pas possible d'interagir avec directement. Si vous souhaitez passer au morceau suivant de votre playlist, vous devez la rendre visible en la plaçant au premier plan.

L'application Liste de courses

Afin d'examiner plus précisément les concepts de vues et de gestionnaires d'affichage, construisons une application Liste de courses qui offre à l'utilisateur la possibilité de sélectionner des articles dans une liste organisée de produits.

Extension Web Vous trouverez tous les codes de cette application dans le répertoire `Sources/Exemple/Chapitre13/AndroidStudio/ListeDeCourses`.

Copyright © 2016 Eyrolles.

Cahier des charges

À son lancement, l'application Liste de courses affiche une liste de rubriques sous la forme de boutons cliquables (figure 13-12). Les rubriques représentent les différents types de produits que l'utilisateur est susceptible d'acheter comme des fruits, des légumes, des produits laitiers...

Figure 13-12 *L'application Liste de courses*

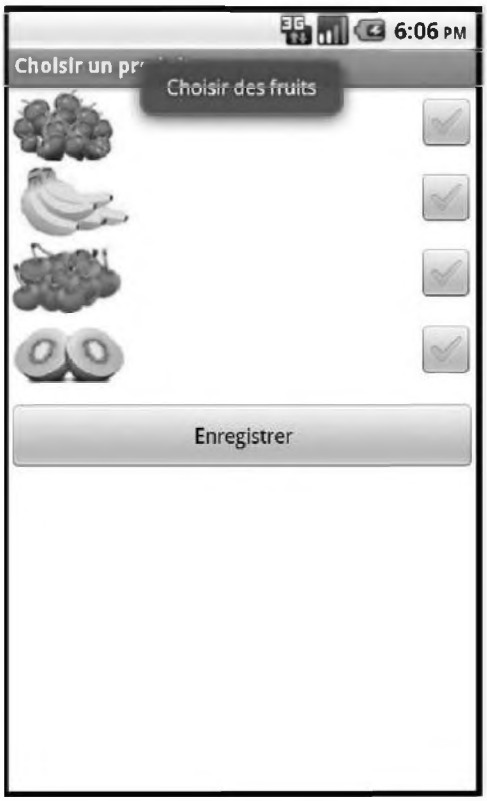


Figure 13-13 *La rubrique Fruits*

Lorsque l'utilisateur sélectionne une rubrique en appuyant sur le bouton associé, un message indique quelle rubrique a été sélectionnée. L'application présente aussi un nouveau panneau, correspondant à la liste des produits de la rubrique sélectionnée (figure 13-13). Il a pour titre « Choisir un produit : ».

L'utilisateur sélectionne ensuite le(s) produit(s) de son choix en cochant la ou les cases associées. Ses choix sont mémorisés lorsqu'il appuie sur le bouton Enregistrer. L'application affiche alors un message indiquant les produits sélectionnés (figure 13-14) et revient à la première vue, soit la liste des rubriques par type de produit.

Copyright © 2016 Eyrolles.

Si l'utilisateur retourne sur une rubrique où il a déjà sélectionné des produits lors d'une étape précédente, la vue correspondant à la liste des produits s'affiche avec les cases des produits sélectionnés cochées.

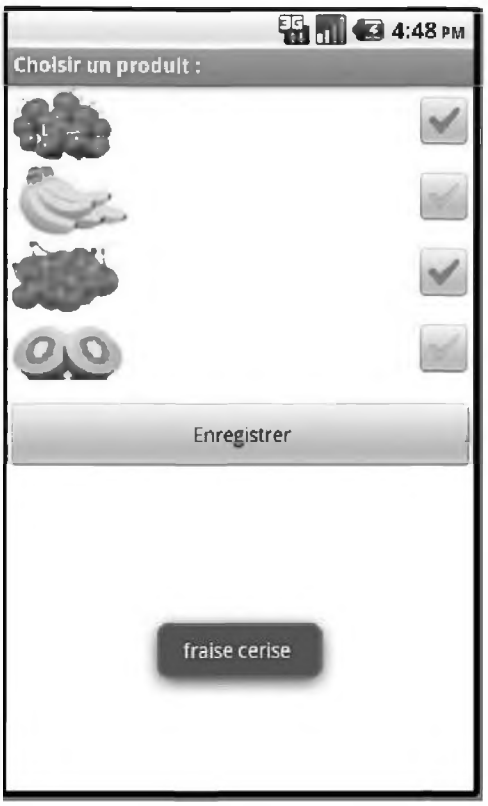


Figure 13-14 Les items Fraise et Cerise sont sélectionnés

Mise en place des éléments graphiques

La première vue affiche une série de boutons correspondant aux différents types de produits de la liste de courses. Ces boutons sont définis au sein du fichier `accueil.xml`, dans le répertoire `Resource/layout` du projet, qui est structuré de la façon suivante :

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
❶ android:background="#FFFFFFFF"
    android:id="@+id/accueil"
```

Copyright © 2016 Eyrolles.

```
>
  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="@drawable/fruit"
    android:id="@+id/fruit"
    android:layout_marginTop="2px"
  />
  <Button
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:background="@drawable/legume"
    android:id="@+id/legume"
    android:layout_marginTop="2px"
  />
  <Button
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
❷ android:text="Laitiers"
    android:id="@+id/laitier"
     android:layout_marginTop="2px"
  />
...
</LinearLayout>
```

La liste des rubriques est définie par une série de boutons gérée par un gestionnaire d'affichage linéaire (`LinearLayout`) qui privilégie l'affichage vertical (`android:orientation`) et fait en sorte d'adapter les éléments qu'il contient à la taille de l'écran (`android:layout_width` et `android:layout_height`).

❶ De nouveaux attributs, `android:background` et `android:id`, apparaissent au sein des balises `LinearLayout` et `<Button>`.

L'attribut `background` est utilisé pour colorier le fond de l'objet qui l'utilise. Ici, l'application a un fond blanc, puisque l'attribut `background` de la balise `LinearLayout` est initialisé à `"#FFFFFFFF"`. Pour les deux premiers boutons, le fond est occupé par une image stockée dans le répertoire `drawable` et non rempli par une couleur. Ici, deux fichiers images nommés `fruit.png` et `legume.png` sont stockés dans le répertoire `Resource/drawable` du projet. Le premier bouton est représenté par une image de fruits, alors que le second l'est avec une image de légumes.

L'attribut `id` est très utile puisqu'il permet de différencier et d'identifier chaque composant défini au sein du fichier XML. Cet identifiant est ensuite utilisé dans le code Java de l'application, où les actions à réaliser seront différentes selon que l'utilisateur appuie sur le bouton dont l'`id` est `fruit` ou `legume`.

Copyright © 2016 Eyrolles.

❷ Un bouton peut ne pas être colorié ou occupé par une image. Dans ce cas, il convient de lui donner un label afin d'expliquer à l'utilisateur sa fonction. La balise `android:text` permet d'écrire un texte sur le bouton. Pour notre exemple, le texte `Laitiers` indique à l'utilisateur qu'en cliquant sur ce bouton, il accède à la liste des produits laitiers.

Mise en place des interactions

Après avoir créés et disposés les différents composants de l'application via le fichier `accueil.xml`, nous devons décrire, en langage Java, les actions à réaliser lorsque l'utilisateur sélectionne une rubrique en appuyant sur le bouton correspondant.

Les instructions décrivant ces comportements sont à insérer dans la classe `Main` comme suit :

```
public class Main extends AppCompatActivity implements
OnClickListener {
     // ❶ Définir des constantes pour distinguer les rubriques
     private static final int FRUIT=0;
     private static final int LEGUME=1;
     private static final int LAITIER=2;
     private static final int BOULANGERIE=3;
     private static final int POISSONNERIE=4;
     private static final int BOUCHERIE=5;
     private static final int CHARCUTERIE=6;
     private static final int ENTRETIEN=7;
    // ❷ Les boutons sont stockés dans un tableau
    public Button [] listeBoutons = new Button[8];
    // À la création de l'application :
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // ❸ Afficher les composants graphiques définis dans le
        // fichier accueil.xml
        setContentView(R.layout.accueil);
        // ❹ Stocker les boutons dans le tableau en fonction de
        // l'id défini dans le fichier accueil.xml
        listeBoutons[FRUIT]=
                    ((Button)this.findViewById(R.id.fruit));
        listeBoutons[LEGUME]=
                    ((Button)this.findViewById(R.id.legume));
        listeBoutons[LAITIER]=
                    ((Button)this.findViewById(R.id.laitier));
        listeBoutons[BOULANGERIE]=
                    ((Button)this.findViewById(R.id.boulangerie));
```

Copyright © 2016 Eyrolles.

```
        listeBoutons[POISSONNERIE]=
                    ((Button)this.findViewById(R.id.poissonnerie));
        listeBoutons[BOUCHERIE]=
                    ((Button)this.findViewById(R.id.boucherie));
        listeBoutons[CHARCUTERIE]=
                    ((Button)this.findViewById(R.id.charcuterie));
        listeBoutons[ENTRETIEN]=
                    ((Button)this.findViewById(R.id.entretien));
        // ❺ Chaque bouton écoute l'événement onClick
        for (int i=0; i < listeBoutons.length; i++) {
            listeBoutons[i].setOnClickListener(this);
        }
    }
     // Action à réaliser lorsqu'un événement onClick est entendu
    public void onClick(View v) {
        String msg="";
        int activite=FRUIT;
        // ❻ Tester l'identifiant de l'objet ayant capturer
        // l'événement onClick et agir en conséquence
        switch (v.getId()) {
            case R.id.fruit :
                msg="Choisir des fruits";
                activite=FRUIT;
                break;
            case R.id.legume :
                msg="Choisir des légumes";
                activite=LEGUME;
                break;
            case R.id.laitier :
                msg="Choisir des produits laitiers";
                activite=LAITIER;
                break;
            case R.id.boulangerie :
                msg="Choisir dans la boulangerie";
                activite=BOULANGERIE;
                break;
            case R.id.poissonnerie:
                msg="Choisir dans la poissonnerie";
                activite=POISSONNERIE;
                break;
            case R.id.boucherie :
                msg="Choisir dans la boucherie";
                activite=BOUCHERIE;
                break;
```

Copyright © 2016 Eyrolles.

```
            case R.id.charcuterie :
                msg="Choisir dans la charcuterie";
                activite=CHARCUTERIE;
                break;
            case R.id.entretien :
                msg="Choisir des produits d'entretien";
                activite=ENTRETIEN;
                break;
        }
        // ❼ Afficher le message correspondant au bouton sélectionné
        Toast msgT = Toast.makeText(this, msg, Toast.LENGTH_SHORT);
         msgT.show();
}
```

La classe `Main` est l'activité (`extends AppCompatActivity`) principale de l'application. Elle est capable de traiter des événements de type `onClick` (`implements OnClickListener`).

❶ À chaque rubrique est associé un numéro mémorisé grâce à une constante de type `final static int`. Chaque constante porte le nom de sa rubrique afin de faciliter la lecture du code et d'éviter des erreurs d'étourderie comme celle d'associer un numéro à un bouton avec une autre action. Il y a huit rubriques différentes, numérotées de 0 à 7.

❷ Pour simplifier le code, les boutons sont stockés dans un tableau composé de huit cases.

❸ La première instruction importante pour notre application consiste à afficher les composants définis au sein du fichier `acceuil.xml`. L'accès à ce fichier s'effectue par l'intermédiaire de la variable `R.layout.acceuil` générée par Android Studio au moment de la compilation.

❹ Nous avons vu à la section précédente, « Mise en place des éléments graphiques », que chaque bouton est identifié par un `id` défini par l'attribut `android:id`. La méthode `findViewById()` est utilisée pour récupérer le composant correspondant à celui passé en paramètre. Ainsi, en passant le paramètre `R.id.fruit` à la méthode `findViewById()`, l'application récupère l'adresse du bouton qui traite la rubrique des fruits.

❺ Lorsque le tableau `listeBoutons` est complet et que toutes les adresses des rubriques sont associées aux indices correspondants, il est temps de parcourir l'intégralité du tableau et de mettre en place un écouteur d'événements pour chaque bouton, via la méthode `setOnClickListener()` qui est appliquée à chaque bouton du tableau `listeBoutons` grâce à la boucle `for`. Ainsi, lorsque l'utilisateur appuie sur un des huit boutons de l'application, un événement `onClick` est émis et la méthode `onClick()` est exécutée.

Copyright © 2016 Eyrolles.

❻ La méthode `onClick()` est exécutée quel que soit le bouton sélectionné. Pour réaliser l'action adéquate, il est nécessaire de déterminer quel bouton a été choisi. Cette vérification est réalisée grâce à l'objet `v` de type `View` passé en paramètre de la méthode `onClick()`. En effet, lorsqu'un événement est émis, la méthode `onClick()` est capable de connaître l'objet qui a capturé l'événement, en utilisant l'expression `v.getId()`. Cette expression contient l'identifiant de l'objet sélectionné, celui défini dans le fichier `accueil.xml`.

Connaissant l'identifiant du composant sélectionné, la structure `switch case` permet d'indiquer à l'application quelle action mener. Suivant la valeur obtenue par l'expression `v.getId()`, le programme mémorise deux valeurs au sein des variables `msg` et `activite`. La première contient le message à afficher confirmant à l'utilisateur que la rubrique qu'il a sélectionnée est bien celle qui correspond au bouton sur lequel il a appuyé. La seconde stocke la constante associée à la rubrique sélectionnée. Elle sera utilisée pour afficher la liste des produits associée à la rubrique (voir section « Créer une seconde activité » ci-après).

❼ La classe `Toast` est à utiliser lorsque vous souhaitez afficher un message éphémère sur l'écran de votre application (figure 13-12, message « Choisir des fruits »). La création du texte et de la bulle qui l'entoure est réalisée par la fonction `makeText()`. Cette méthode demande en premier paramètre une valeur correspondant au contexte de l'application (ici `this`, c'est-à-dire l'application elle-même). Le message à afficher est placé en deuxième paramètre. Le troisième paramètre indique combien de temps ce message doit rester affiché (puisqu'il est éphémère). Deux valeurs sont possibles : `Toast.LENGTH_SHORT` ou `Toast.LENGTH_LONG`.

Créer une seconde activité

Une fois la rubrique sélectionnée, un nouveau panneau s'affiche présentant la liste des produits correspondant à la rubrique choisie. Ce panneau est en réalité une nouvelle activité créée à partir de l'activité principale.

La création de cette nouvelle activité s'effectue en trois étapes.

1. Mise en place des composants graphiques (voir section suivante « Les éléments graphiques »).
2. Écriture d'une activité au sein d'une nouvelle classe (voir section suivante « L'activité RubriqueFruit »).
3. Lancement de la nouvelle activité à partir de l'activité principale (voir section suivante « Lancer une seconde activité »).

Remarque

Vous réaliserez la rubrique « Légumes » en vous reportant à la section « Exercices » située à la fin de ce chapitre.

Copyright © 2016 Eyrolles.

Les éléments graphiques

Les composants graphiques et leur agencement sont définis au sein du fichier `fruits.xml` que vous trouverez dans le répertoire `Resource/layout`. La disposition des composants est un peu plus complexe que celle des exemples précédents, car elle nécessite d'imbriquer les layouts. Examinons plus précisément l'extrait du fichier `fruits.xml` suivant :

```
<LinearLayout ❶
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#FFFFFFFF"
>
  <LinearLayout ❷
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:orientation="horizontal"
   android:background="#FFFFFFFF"
  >
    <ImageView
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:src="@drawable/fraise"
     android:layout_marginTop="3px"
    />
    <CheckBox
     android:layout_height="wrap_content"
      android:id="@+id/fraise"
      android:layout_width="wrap_content"
      android:layout_gravity="right"
      android:layout_marginTop="3px"
    />
  </LinearLayout> ❷

  <LinearLayout ❸
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:orientation="horizontal"
   android:background="#FFFFFFFF"
  >
    <ImageView
     android:layout_width="wrap_content"
```

Copyright © 2016 Eyrolles.

```
        android:layout_height="wrap_content"
        android:src="@drawable/banane"
        android:layout_marginTop="3px"
       />
       <CheckBox
        android:layout_height="wrap_content"
         android:id="@+id/banane"
         android:layout_width="wrap_content"
         android:layout_gravity="right"
         android:layout_marginTop="3px"
       />
    </LinearLayout> ❸
      .....
    <Button
     android:layout_width="fill_parent"
     android:layout_height="wrap_content"
     android:text="Enregistrer"
     android:id="@+id/Save"
     android:layout_marginTop="10px"
    />
</LinearLayout> ❶
```

❶ Le premier layout (figure 13-14, repère ❶) gère l'affichage des composants qu'il contient de façon verticale (`android:orientation="vertical"`). Il est constitué de cinq éléments : quatre layouts (dans l'extrait, deux sont présentés) et un bouton, qui sont donc placés les uns après les autres dans le sens de la hauteur. Ce layout englobe ces cinq composants car la balise fermante `</LinearLayout>` correspondant à la balise de création du layout se situe en toute fin du fichier (voir les deux repères ❶ de l'extrait de code).

❷ Le deuxième layout (figure 13-14, repère ❷) gère l'affichage des composants qu'il contient de façon horizontale (`android:orientation="horizontal"`). Il est constitué de deux éléments : `ImageView` et `CheckBox` qui sont placés l'un à la suite de l'autre dans le sens de la largeur. Ce layout est imbriqué dans le premier layout car la balise fermante `</LinearLayout>` correspondant à la balise de création du layout se situe juste après la définition des composants qu'il contient (voir les deux repères ❷ de l'extrait de code).

❸ Le troisième layout (figure 13-15, repère ❸) gère également l'affichage des composants qu'il contient de façon horizontale (`android:orientation="horizontal"`). Il est également constitué de deux éléments : `ImageView` et `CheckBox`.

Copyright © 2016 Eyrolles.

Remarque Le composant `ImageView` est utilisé pour afficher une image alors que `CheckBox` est un composant à deux états dont on peut vérifier par programme s'il est sélectionné ou non.

Les layouts ❷ et ❸ sont identiques dans leur agencement et leurs composants graphiques. Seul le contenu des `ImageView` (`android:src="@drawable/fraise"` et `android:src="@drawable/banane"`) et l'identifiant (`android:id="@+id/fraise"` et `android:src="@+id/banane"`) permettent de les différencier.

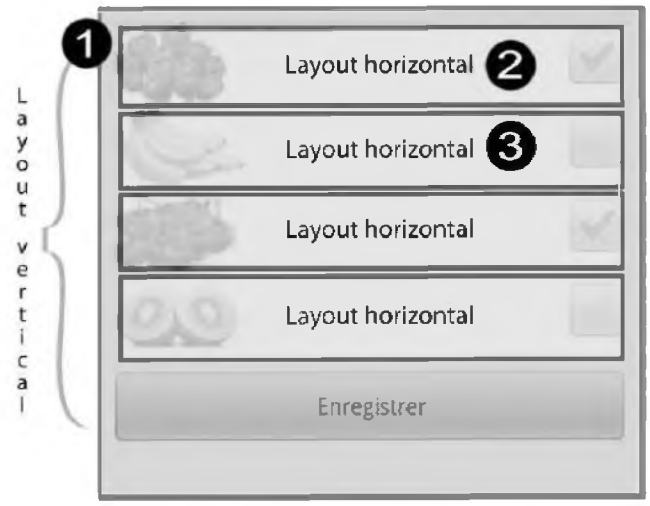


Figure 13-15 *Imbrication de layouts*

L'activité RubriqueFruits

Après avoir « dessiné » l'activité correspondant à la rubrique « Fruits », nous devons créer la classe `RubriqueFruits` associée de la façon suivante :

```
public class RubriqueFruits extends AppCompatActivity
                            implements View.OnClickListener{
public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.fruits);
   }
}
```

L'activité `RubriqueFruits` est pour l'instant très simple. Elle contient, comme toute activité, la méthode `onCreate()` qui appelle la fonction `setContentView()`. Cette méthode affiche les composants graphiques définis au sein du fichier dont le nom est passé en paramètre, c'est-à-dire ici le fichier `fruits.xml`.

Copyright © 2016 Eyrolles.

Lancer une seconde activité

Le panneau contenant la liste des fruits s'affiche lorsque l'utilisateur appuie sur le bouton Fruits de l'activité principale `Main`. L'activité `RubriqueFruits` est donc lancée à partir de la méthode `onClick()`, immédiatement après l'affichage du message indiquant quelle rubrique a été sélectionnée.

```
public void onClick(View v) {
   // Code identique à celui présenté à la section « Mise en place
   // des interactions »
   Toast msgT = Toast.makeText(this, msg, Toast.LENGTH_SHORT);
   msgT.show();
   // ❶ Créer une nouvelle activité en fonction de la rubrique
   // sélectionnée
   creerActivite(activite);
}
// Créer une activité en fonction de la valeur passée en paramètre
public void creerActivite(int tmp){
    Intent nvActivite;
    switch (tmp) {
        case FRUIT :
          // ❷ Création de l'Intent
         nvActivite = new Intent(Main.this, RubriqueFruits.class);
          startActivityForResult(nvActivite, FRUIT);
        break;
    }
}
```

❶ Pour simplifier la lecture du code, nous insérons à la fin de la méthode `onClick()` l'appel à une nouvelle fonction `creerActivite()` qui a pour rôle (comme son nom l'indique) de créer une nouvelle activité. L'activité produite est différente selon la valeur fournie en paramètre.

❷ À l'appel de la fonction `creerActivite()`, un objet `nvActivite` est créé de type `Intent`. Les `Intent` ont été conçus pour faciliter la vie du développeur d'applications Android. Ils offrent la possibilité de lancer un programme (avec ou sans interface graphique) à partir d'un premier programme, tout en gardant un canal de communication entre les deux.

Le type de l'activité est créé en fonction de la valeur passée en paramètre. Cette valeur fait l'objet d'un test au sein d'une structure `switch case`. Dans le cas où le paramètre correspond à la constante `FRUIT`, la création de l'`Intent` s'effectue alors grâce à l'instruction `new Intent(Main.this, RubriqueFruits.class)`. Les paramètres du constructeur `Intent` indiquent quels programmes sont à mettre en relation. Le second paramètre précise le programme à lancer (ici, `RubriqueFruits.class`).

Attention, la création d'un `Intent` ne suffit pas au bon fonctionnement de votre application. Vous devez obligatoirement déclarer la nouvelle activité `RubriqueFruits` dans le fichier

Copyright © 2016 Eyrolles.

AndroidManifest.xml. En cas d'oubli, vous obtiendrez une erreur de compilation. Cette déclaration s'effectue comme suit :

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.ListeDeCourses"
  android:versionCode="1"
  android:versionName="1.0">
  <application android:label="@string/app_name">
   <activity android:name="Main"
    android:label="@string/app_name">
    <intent-filter>
       <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
     </intent-filter>
    </activity>
    <activity
      android:name=".RubriqueFruits"
      android:label="Choisir un produit :"
    />
  </application>
</manifest>
```

L'attribut android:name indique le nom de la nouvelle activité à prendre en compte, tandis que android:label précise le titre à afficher pour cette nouvelle activité.

Remarque

Le fichier AndroidManifest.xml se trouve dans le répertoire manifests de l'arborescence de tout projet Android Studio.

Quelques outils incontournables

L'activité « Choisir un produit : » s'affiche à présent lorsque l'utilisateur sélectionne une des rubriques présentées par l'activité Liste de courses (figure 13-13). L'objectif est maintenant d'écrire le cœur de métier de la seconde activité à savoir :

- afficher un message indiquant quels produits ont été sélectionnés (voir section suivante « Les cases à cocher ») ;
- mémoriser ces produits afin de connaître à tout moment quel produit a été coché (voir section suivante « Stocker les données dans un fichier texte ») ;
- effacer l'activité « Choisir un produit : » lorsque l'utilisateur a enregistré ses produits et revenir à l'activité principale (voir section suivante « Effacer l'activité en cours ») ;
- si l'utilisateur retourne à une rubrique où des produits ont déjà été sélectionnés, les cases correspondantes doivent être cochées (voir sections « Lire des données depuis un fichier texte » et « Extraire des données à partir d'une chaîne de caractères »).

Copyright © 2016 Eyrolles.

Les cases à cocher

Pour vérifier l'état des cases à cocher, nous devons créer des objets de type CheckBox en les déclarant comme variables de classe comme suit :

```
public class RubriqueFruits extends AppCompatActivity
                              implements View.OnClickListener {
// Déclaration d'objets de type CheckBox comme variables de classe
private CheckBox chkBanane, chkCerise, chkFraise, chkKiwi;
private Button btnSave;

public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.fruits);
    //  ❶ Associer les CheckBox définies dans le fichier
    //  fruits.xml avec les variables de classe
    chkBanane = (CheckBox) findViewById(R.id.banane);
    chkCerise = (CheckBox) findViewById(R.id.cerise);
    chkFraise = (CheckBox) findViewById(R.id.fraise);
    chkKiwi = (CheckBox) findViewById(R.id.kiwi);
    btnSave = (Button) findViewById(R.id.Save);
    // ❷ Mettre en place un écouteur d'événements sur le bouton
    // Enregistrer
    btnSave.setOnClickListener(this);
}
// Lorsqu'on clique sur le bouton Enregistrer
public void onClick(View v) {
    Toast toaster;
    String msg="";
   // ❸ Traiter l'état des CheckBox
    if (chkFraise.isChecked())
          msg+="fraise+";
    if (chkBanane.isChecked())
          msg+="banane+";
    if (chkCerise.isChecked())
          msg+="cerise+";
    if (chkKiwi.isChecked())
          msg+="kiwi+";
    // ❹ Afficher un message éphémère s'il n'est pas vide
    if (!msg.equals("")) {
      String msgToast = msg.replace("+", " ");
      toaster = Toast.makeText(this.getApplicationContext(),
                               msgToast, Toast.LENGTH_LONG);
      toaster.show();
    }
}
```

Copyright © 2016 Eyrolles.

❶ La méthode `findViewById()` permet ici d'associer les cases à cocher et le bouton définis au sein du fichier `fruits.xml` avec les objets (`chkBanane`, `chkCerise`, ..., `btnSave`) déclarés et utilisés par la classe `RubriqueFruits.java`. Observez que pour récupérer correctement l'adresse de l'objet, vous devez spécifier son type à l'aide d'un cast `(CheckBox)` ou `(Button)`.

❷ Afin de mémoriser ses choix, l'utilisateur appuie sur le bouton Enregistrer. Un écouteur d'événements est donc ajouté à l'objet `btnSave`. La classe `RubriqueFruits` doit également implémenter `OnClickListener`.

❸ La vérification de l'état des cases à cocher ne peut s'effectuer que lorsque l'utilisateur a appuyé sur le bouton Enregistrer. Ainsi, nous devons placer quatre tests `if` dans la méthode `onClick()`, pour vérifier si les quatre cases à cocher sont sélectionnées ou non. Ces tests utilisent l'expression `objet.isChecked()`, car la méthode `isChecked()` appliquée à un `objet` de type `CheckBox` retourne le booléen `true` lorsque la case est effectivement cochée.

Lorsqu'une case est cochée, la variable `msg` s'incrémente du terme correspondant au produit sélectionné, suivi du signe `+`. Par exemple, lorsque les cases « fraise » et « banane » sont cochées, la variable `msg` contient le texte `fraise+banane+`.

❹ Le signe `+` sera utilisé lors de l'extraction des données à partir du fichier texte (voir section suivante « Lire des données depuis un fichier texte »). Il nuit à la bonne lecture du message éphémère, c'est pourquoi le message `msgToast` affiché par le `Toast` est modifié. Tous les signes `+` contenus dans `msg` sont remplacés par des espaces (`replace("+", " ")`).

Stocker les données dans un fichier texte

Il existe plusieurs techniques pour stocker les données à partir d'une application mobile. Il est ainsi possible de sauvegarder les données sous forme de Préférences (`SharedPreferences`), d'utiliser une base de données de type SQLite, ou encore de recourir au stockage interne sous forme de fichiers standards. Nous avons choisi ici de vous présenter cette dernière technique, car elle est sensiblement plus proche de celle présentée au chapitre 10, « Collectionner un nombre indéterminé d'objets », section « Les fichiers textes ».

Remarque

Les capacités de stockage d'un téléphone mobile restent limitées. Il convient de ne pas en abuser et de restreindre ce mode de conservation de l'information à des données « légères » comme du texte.

L'écriture des produits sélectionnés s'effectue immédiatement après l'affichage du message éphémère décrit à la section précédente « Les cases à cocher ». Les instructions sont les suivantes :

```
public void onClick(View v) {
   // Créer et afficher le message dans un toast (voir section
   // précédente)
```

Copyright © 2016 Eyrolles.

```
    toaster.show();
    // Mémoriser les fruits sélectionnés
    ecrireListeFruits(msg);
}
```

La méthode `ecrireListeFruits()` est appelée pour stocker le message passé en paramètre dans un fichier texte. Les instructions composant cette fonction sont les suivantes :

```
public void ecrireListeFruits(String tmp) {
    FileOutputstream fos;
    try {
      fos = openFileOutput("Fruits.txt", Context.MODE_PRIVATE);
      Log.i("----------  Fichier : ", getFilesDir().toString());
      fos.write(tmp.getBytes());
      fos.close();
    }
    catch (IOException ex){
      Log.i("----------  Fichier : ", "Erreur d'écriture ...");
    }
}
```

Un objet de type `FileOutputstream` est déclaré, puis créé au sein d'une structure `try catch` afin d'éviter les erreurs d'entrée-sortie en cours d'exécution. L'ouverture en écriture du fichier est réalisée par l'intermédiaire de la fonction `openFileOutput()` qui prend en paramètres deux valeurs. Le premier paramètre définit le nom du fichier de sauvegarde (ici, `Fruits.txt`) alors que le second précise le mode d'accès au fichier (ici, `MODE_PRIVATE`). Grâce au mode privé, le contenu du fichier est effacé si le fichier existe déjà et les autres applications n'ont pas accès aux données contenues dans le fichier.

Le message, traduit sous forme d'octets, est passé en paramètre de la méthode `fos.write()` pour être ensuite écrit dans le fichier associé à l'objet `fos`.

Remarque

L'instruction `Log.i()` est utilisée pour afficher des commentaires et/ou le contenu de certains objets dans une fenêtre d'exécution appelée `Log Window` ou `LogCat`. Elle apparaît lorsque l'on sélectionne l'onglet Android Monitor en bas à gauche de l'IDE. L'instruction `log.i()`, placée dans le bloc `try` de notre exemple, affiche le nom du répertoire où est écrit le fichier `Fruits.txt`.

Effacer l'activité en cours

Lorsque l'utilisateur appuie sur le bouton Enregistrer, les données sont stockées dans un fichier texte et l'activité « `Choisir un produit :` » s'efface. L'activité initiale `Liste de courses` réapparaît ; l'utilisateur peut alors sélectionner une nouvelle rubrique. La fermeture d'une activité en cours d'exécution s'effectue tout simplement par l'intermédiaire de la méthode `finish()`, comme suit.

Copyright © 2016 Eyrolles.

```
public void onClick(View v) {
    // Mémoriser les fruits sélectionnés (voir section précédente)
    EcrireListeFruit(msg);
    // Fermer l'activité en cours
    fermerLesFruits();
}
public void fermerLesFruits(){
     this.finish();
}
```

La méthode `fermerLesFruits()` est appelée aussitôt après avoir enregistré les données dans le fichier `Fruits.txt`. Elle contient une seule instruction `this.finish()` qui ferme l'activité en cours.

Lire des données depuis un fichier texte

Si l'utilisateur retourne sur une rubrique, les cases correspondant aux produits qu'il a sélectionnés doivent apparaître cochées. Pour réaliser cette opération, nous utilisons les données stockées dans le fichier `Fruits.txt`, afin de cocher par programme les cases correspondant aux fruits mémorisés.

La première étape consiste à récupérer le texte stocké dans le fichier `Fruits.txt`, comme suit :

```
public String lireListeFruits() {
    FileInputstream fis;
    String  data="";
    try {
      fis= openFileInput("Fruits.txt");
      char[] charLus = new char[255];
      InputstreamReader isr = new InputstreamReader(fis);
      isr.read(charLus);
      data = new String(charLus);
    }
    catch (IOException ex){
      Log.i("----------  Fichier : ", "Erreur de lecture ...");
    }
    return data;
}
```

Un objet de type `FileInputstream` est déclaré, puis créé au sein d'une structure `try catch` afin d'éviter les erreurs d'entrée-sortie. L'ouverture en lecture du fichier est réalisée par l'intermédiaire de la fonction `openFileInput()` qui prend en paramètre le nom du fichier à lire (ici, `Fruits.txt`). Après ouverture, le fichier est lu par l'intermédiaire d'un flux

Copyright © 2016 Eyrolles.

de lecture (`InputstreamReader`). Les données lues (`charLus`) sont ensuite converties en chaîne de caractères (`data`).

La méthode `lireListeFruits()` est appelée dès la création de l'activité `RubriqueFruits`, comme suit :

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.fruits);
    String listeFruits = lireListeFruits();
    // Création et initialisation des CheckBox (voir le code de la
    // section « Les cases à cocher », repère ❶)
```

À l'appel de la méthode, le fichier est lu et le message extrait est placé en retour dans la chaîne `listeFruits`.

Extraire des données à partir d'une chaîne de caractères

Une fois que le message est lu, nous devons extraire les termes correspondant aux noms des produits sélectionnés. Cette extraction passe par l'utilisation de la classe prédéfinie `StringTokenizer`.

Pour en savoir plus L'extraction de données d'un fichier texte est étudiée à la section « Exercices – Créer des fichiers textes – Exercice 10.5 » du chapitre 10, « Collectionner un nombre indéterminé d'objets ».

Grâce aux outils de la classe `StringTokenizer`, il est possible d'extraire des mots d'un message si ce dernier est constitué de mots séparés par un caractère spécifique appelé séparateur. Dans notre exemple, chaque mot (représentant un produit) est séparé par un signe +. L'extraction des produits enregistrés et la mise à jour des cases à cocher correspondantes s'effectuent alors, comme suit :

```
public void majCheckBox(String tmp) {
   // Créer un objet st qui détecte des champs de mots et des
   // séparateurs "+"
   StringTokenizer st = new StringTokenizer(tmp,"+");
   int i=0;
   // Créer un tableau dont la longueur correspond au nombre de
   // champs séparés par des "+"
   String mot[] = new String[st.countTokens()];
   // Tant qu'il y a des champs séparés par des "+"
   while (st.hasMoreTokens()) {
     // Enregistrer le champ courant dans le tableau mot à l'indice i
     mot[i] = st.nextToken();
      if (mot[i].equals("banane")) {
        // Si mot[i] vaut "banane", cocher la case correspondante
        chkBanane.setChecked(true);
      }
```

Copyright © 2016 Eyrolles.

```
        else if (mot[i].equals("fraise")) {
          // Si mot[i] vaut "fraise", cocher la case correspondante
          chkFraise.setChecked(true);
        }
        else if (mot[i].equals("cerise")) {
           // Si mot[i] vaut "cerise", cocher la case correspondante
           chkCerise.setChecked(true);
        }
        else if (mot[i].equals("kiwi")){
          // Si mot[i] vaut "kiwi", cocher la case correspondante
          chkKiwi.setChecked(true);
        }
        i++;
      }
  }
```

La méthode `majCheckBox()` parcourt l'ensemble du message passé en paramètre et extrait chaque terme se situant après le séparateur +. Une fois les termes extraits, elle vérifie s'ils correspondent à un nom de produit. Si tel est le cas, elle modifie l'état de la case à cocher (`setChecked(true)`) de façon à ce qu'elle apparaisse effectivement cochée.

La méthode `majCheckBox()` est appelée après la création des cases à cocher, comme suit :

```
public void onCreate(Bundle savedInstanceState) {
    // …
    String listeFruits = lireListeFruits();
    // Création et initialisation des CheckBox (voir le code de la
    // section « Les cases à cocher », repère ❶)
    chkBanane = (CheckBox) findViewById(R.id.banane);
    // …
    majCheckBox(listeFruits);
```

Le message traité par la méthode `majCheckBox()` est celui extrait du fichier `Fruits.xml`, récupéré en résultat de la fonction `lireListeFruits()`.

Publier une application Android

Une fois votre application correctement développée, il est nécessaire de vérifier son bon fonctionnement directement sur votre mobile. Ceci est d'autant plus important qu'il est impossible sur l'émulateur de traiter le *multi-touch* ou de détecter l'inclinaison du mobile. À la lecture de la section « Tester votre application sur un mobile Android », vous apprendrez à tester vos applications en cours de développement, directement sur votre téléphone.

Copyright © 2016 Eyrolles.

Une fois votre application Android développée, vous souhaiterez sans doute la distribuer sur Internet, en la vendant ou en en faisant don. Pour cela, vous devez la publier sur des serveurs dédiés. La plate-forme officielle du marché d'applications Android est l'Android Market, tout récemment renommée Google Play. Reportez-vous à la section « Déposer une application Android sur un serveur dédié » pour découvrir toutes les étapes de publication d'une application.

Tester votre application sur un mobile Android

Pour tester votre application nouvellement développée, vous devez tout d'abord configurer votre mobile comme suit.

1. Allumez votre portable et appuyez sur l'icône Paramètres sur l'écran général. Le panneau propose plusieurs rubriques, celles qui nous intéressent ici sont Sécurité et Options de développement (figure 13-16).

Figure 13-16 *Le panneau Paramètres*

2. Appuyez sur la rubrique Options de développement afin d'ouvrir le panneau correspondant. Différentes options sont disponibles ; activez l'option Débogage USB (figure 13-17).

Figure 13-17 *Le panneau Options de développement*

Copyright © 2016 Eyrolles.

Remarque Avec la version 4.3 d'Android, l'accès aux Options de développement n'apparaît que lorsque vous avez, avant tout, affiché la page « À propos de l'appareil » de la rubrique Paramètres, onglet Plus. Sur l'écran « À propos de l'appareil », descendez à la ligne Numéro de version. Tapez sept fois sur cette ligne. L'option Options de développement s'affiche maintenant dans l'onglet Plus de vos Paramètres.

3. Un panneau de confirmation apparaît. Appuyez sur OK pour confirmer l'activation de cette option (figure 13-18).

Figure 13-18 *Confirmer l'autorisation de débogage*

4. Revenez au panneau Paramètres et appuyez sur la rubrique Sécurité (figure 13-16). Dans le panneau Sécurité, activez l'option Sources inconnues (figure 13-19).

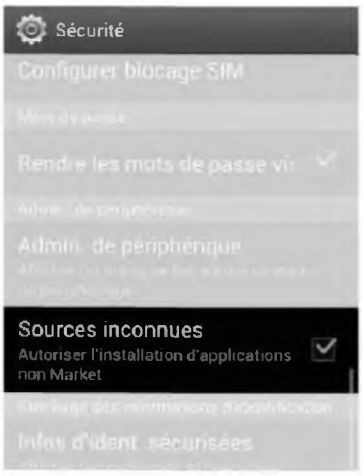


Figure 13-19 *Le panneau Sécurité*

Copyright © 2016 Eyrolles.

5. Un panneau d'alerte apparaît, indiquant que vous êtes responsable du bon ou mauvais fonctionnement des applications que vous développez. Appuyez sur OK pour confirmer le choix de cette option (figure 13-20).

Figure 13-20 *Confirmer l'autorisation d'utiliser des applications non signées*

Votre téléphone est configuré. Il ne vous reste plus qu'à le brancher à votre ordinateur via le port USB.

Retournez sous Android Studio pour exécuter le projet relatif à l'application que vous souhaitez tester. Le panneau Device Chooser apparaît ; sélectionnez la ligne correspondant à votre portable et cliquez sur OK (figure 13-21). Votre application s'affiche sur votre mobile ; vous n'avez plus qu'à tester son bon fonctionnement.

Figure 13-21 *Le panneau Device Chooser*

Copyright © 2016 Eyrolles.

Déposer une application Android sur un serveur dédié

Pour mettre à disposition votre application sur Internet, vous devez réaliser un certain nombre d'opérations. Commencez par vous inscrire sur la plate-forme Android (voir section suivante « Créer un compte développeur »), puis transformez votre projet Android Studio en application Android à part entière (voir section suivante « Passer du projet à l'application »). Une fois cela effectué, vous pourrez déposer l'application sur le site Google Play (voir section suivante « Déposer une application »).

Créer un compte développeur

Pour créer un compte développeur, rendez-vous sur le site *http://play.google.com/apps/publish*, en vous connectant obligatoirement avec votre adresse Gmail.

Pour créer votre profil développeur, vous devez compléter le formulaire de la page d'accueil (figure 13-22). Une fois les champs correctement remplis, cliquez sur Poursuivre et payer.

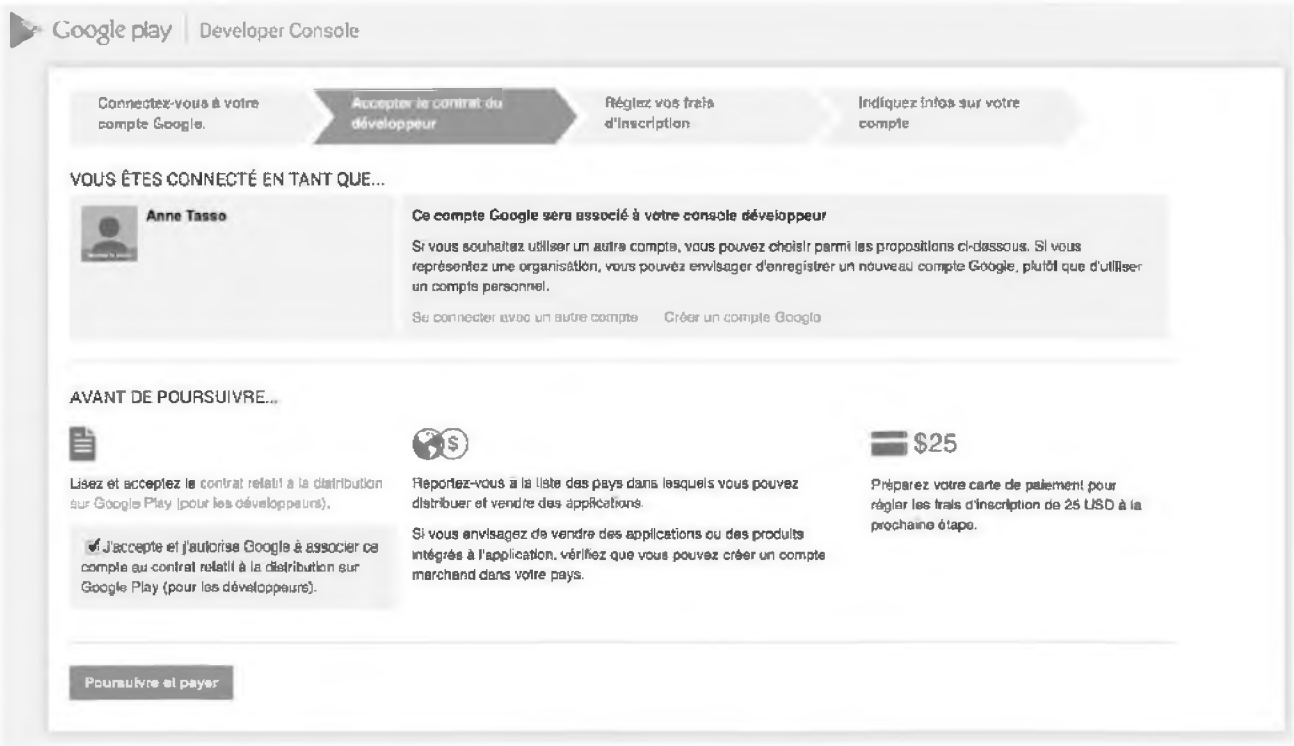


Figure 13-22 *Page d'accueil de création d'un compte développeur*

Copyright © 2016 Eyrolles.

À la page suivante (figure 13-23), acceptez le contrat après avoir lu ses termes, puis cliquez sur « J'accepte. Continuer ».

Figure 13-23 *Contrat pour les développeurs*

Copyright © 2016 Eyrolles.

L'inscription à un compte développeur Android est payante. Les frais s'élèvent à 25 dollars américains (environ 20 €). Ils ne sont demandés qu'une seule fois et vous donnent le droit d'accéder à la plate-forme de publication de vos applications. Pour régler ces frais d'inscription, il vous suffit de cliquer sur le bouton Accepter et continuer (figure 13-24) après avoir rempli le formulaire précisant vos coordonnées bancaires.

Figure 13-24 *Frais d'inscription*

Copyright © 2016 Eyrolles.

Après paiement et après validation de vos données par les services de Google, vous arrivez sur la page qui vous permet de publier vos applications (figure 13-25).

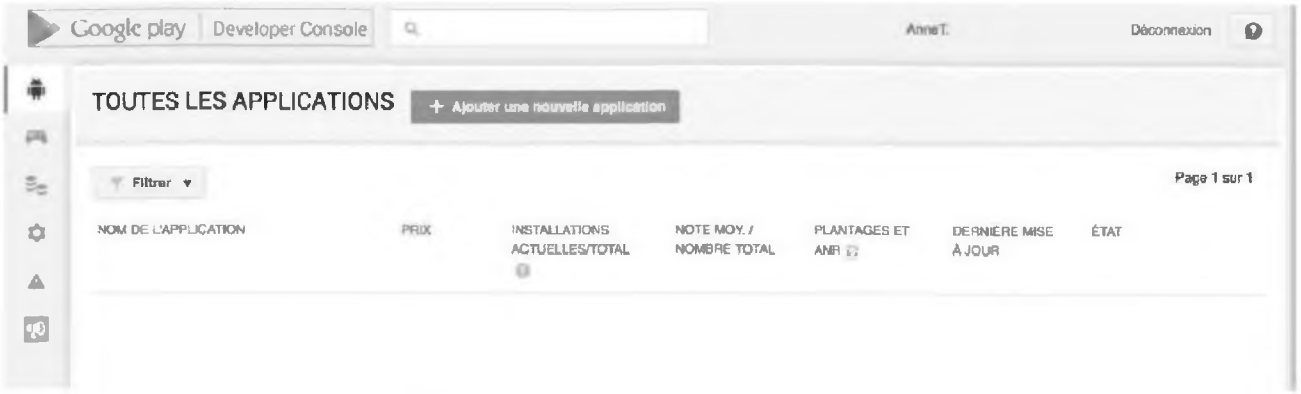


Figure 13-25 *Console de publication des applications Android*

Passer du projet à l'application

Les applications Android ont été développées et exécutées sous forme de projets Android Studio. Pour que ces projets deviennent des applications à part entière, il faut les transformer en applications de type `nomApplication.apk` (`apk` pour *android package*).

Cette transformation s'effectue en plusieurs étapes décrites ci-après.

Définir une icône pour l'application

Toute application Android possède une icône qui l'identifie. Présente sur l'écran de votre mobile, elle permet de lancer l'application lorsque vous appuyez dessus.

Par défaut, toutes les applications développées par Android Studio possèdent une icône représentant le petit androïde vert caractéristique des systèmes Android. Pour donner une identité visuelle à vos applications, il est conseillé de leur associer une icône spécifique. La marche à suivre est la suivante.

1. Créez l'icône au format PNG. La taille de l'icône est de 96 × 96 pixels en moyenne résolution. Le plus souvent, cette image sera nommée `icon.png`. Placez le fichier dans le répertoire `res/drawable` du projet Android Studio.
2. Dans le fichier `AndroidManifest.xml`, insérez la ligne suivante :

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.ListeDeCourses"
  android:versionCode="1"
  android:versionName="1.0">
```

Copyright © 2016 Eyrolles.

```
    <uses-sdk android:minSdkVersion="8"/>
    <application
          android:label="@string/app_name"
          android:icon="@drawable/icon"
      >
     <activity android:name="Main"
       ...
    </application>
</manifest>
```

3. Vérifiez la présence de la nouvelle icône en lançant l'application directement sur votre mobile via le port USB. L'icône doit être visible dans la liste des applications de votre mobile.

Créer une signature

Toute application publiée sur Google Play doit être certifiée afin de garantir son bon usage et éviter la propagation de programmes malveillants. La signature d'une application passe par la création d'une clé :

1. Sélectionnez l'item Generate Signed APK... du menu Build (figure 13-26).

Figure 13-26 *Exporter et signer une application*

Copyright © 2016 Eyrolles.

2. Le panneau Generate Signed APK apparaît (figure 13-27, ❶), cliquez sur Next. Cliquez ensuite sur le bouton Create new… (figure 13-27, ❷).

Figure 13-27 *Le panneau Generate Signed APK*

Le panneau New Key Store s'affiche (figure 13-28).

1. Dans le champ Key store path, saisissez le chemin d'accès du répertoire où sera stocké le certificat ou cliquez sur le bouton … pour le localiser sur votre ordinateur.

Figure 13-28 *Ajouter une nouvelle clé*

Copyright © 2016 Eyrolles.

2. Entrez votre mot de passe dans le champ Password et saisissez-le une seconde fois dans le champ Confirm pour le valider. Veillez à bien mémoriser les mots de passe car ils vous seront demandés à chaque nouvelle version de votre application.
3. Saisissez un nom d'Alias ainsi qu'un mot de passe.
4. Pour finir, entrez toutes les informations demandées pour le certificat (Nom, Prénom…). Ces informations sont nécessaires à la création de la clé, puis cliquez sur le bouton OK.
5. Le panneau Generate Signed APK réapparaît avec tous les champs remplis (figure 13-29), cliquez sur Next.

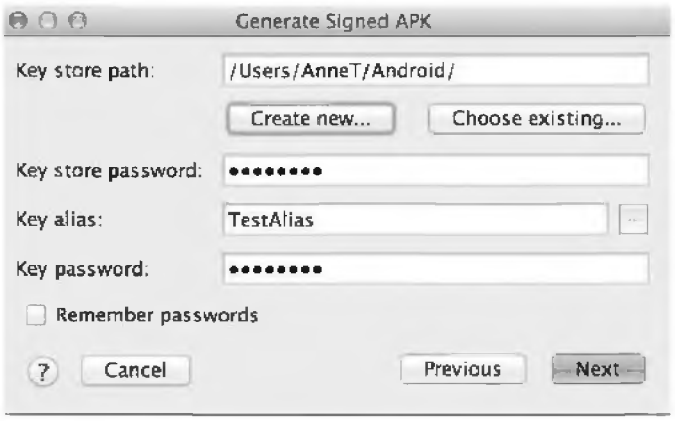


Figure 13-29 *La clé est créée.*

6. La dernière étape consiste à indiquer l'emplacement de l'application finale (figure 13-30), c'est-à-dire celle que nous allons ensuite déposer sur le serveur Google Play. Cliquez sur Finish pour valider la création de l'APK. Le projet est compilé et devient une application autonome.

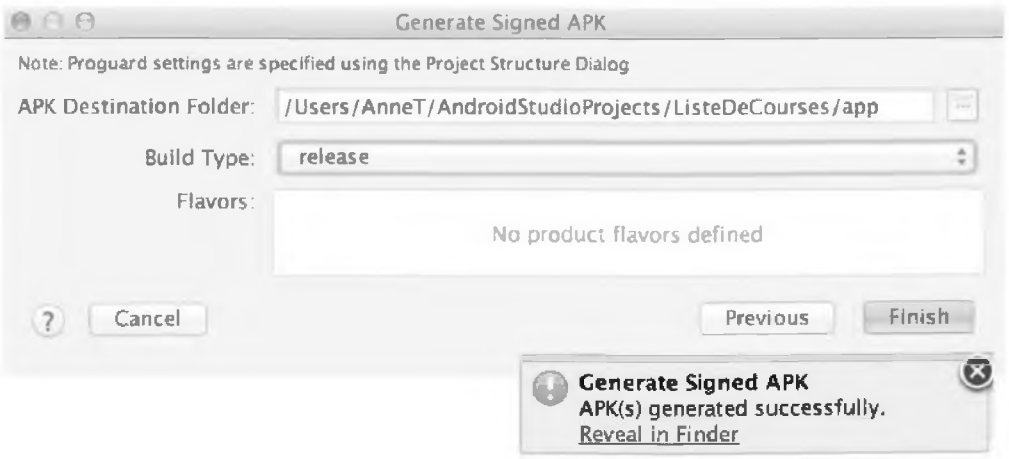


Figure 13-30 *Déterminer l'emplacement final de l'APK*

Copyright © 2016 Eyrolles.

Déposer une application

Le dépôt d'une application s'effectue sur la plate-forme Google Play, après création de votre compte développeur et d'une clé pour l'application (voir section précédente « Création d'un compte développeur »). La démarche de publication est la suivante :

1. Sur la page de votre compte développeur Android, cliquez sur le bouton Ajouter une nouvelle application (figure 13-25).
2. Un formulaire apparaît vous proposant de nommer votre application, d'importer le fichier APK de votre application (figure 13-31, repère ❶) et de préparer la fiche d'informations relative à votre application (figure 13-31, repère ❷). Les deux opérations peuvent s'effectuer dans n'importe quel ordre. Le formulaire vous demande également de choisir la langue par défaut de votre application. Attention, choisissez la bonne langue, cette donnée ne pourra plus être modifiée par la suite.

Figure 13-31 *Donner un nom à votre application*

Copyright © 2016 Eyrolles.

3. En cliquant sur le bouton Importer un fichier APK, une nouvelle page s'affiche constituée de rubriques permettant la navigation entre l'import de l'application (figure 13-32, repère ❶) et la définition des données la concernant (figure 13-32, repère ❷). Si vous cliquez sur le bouton Importer votre premier fichier APK…, un panneau apparaît (figure 13-32, repère ❸). Il vous permet soit de glisser-déposer votre application, soit de la rechercher sur votre ordinateur en parcourant votre système de fichiers.

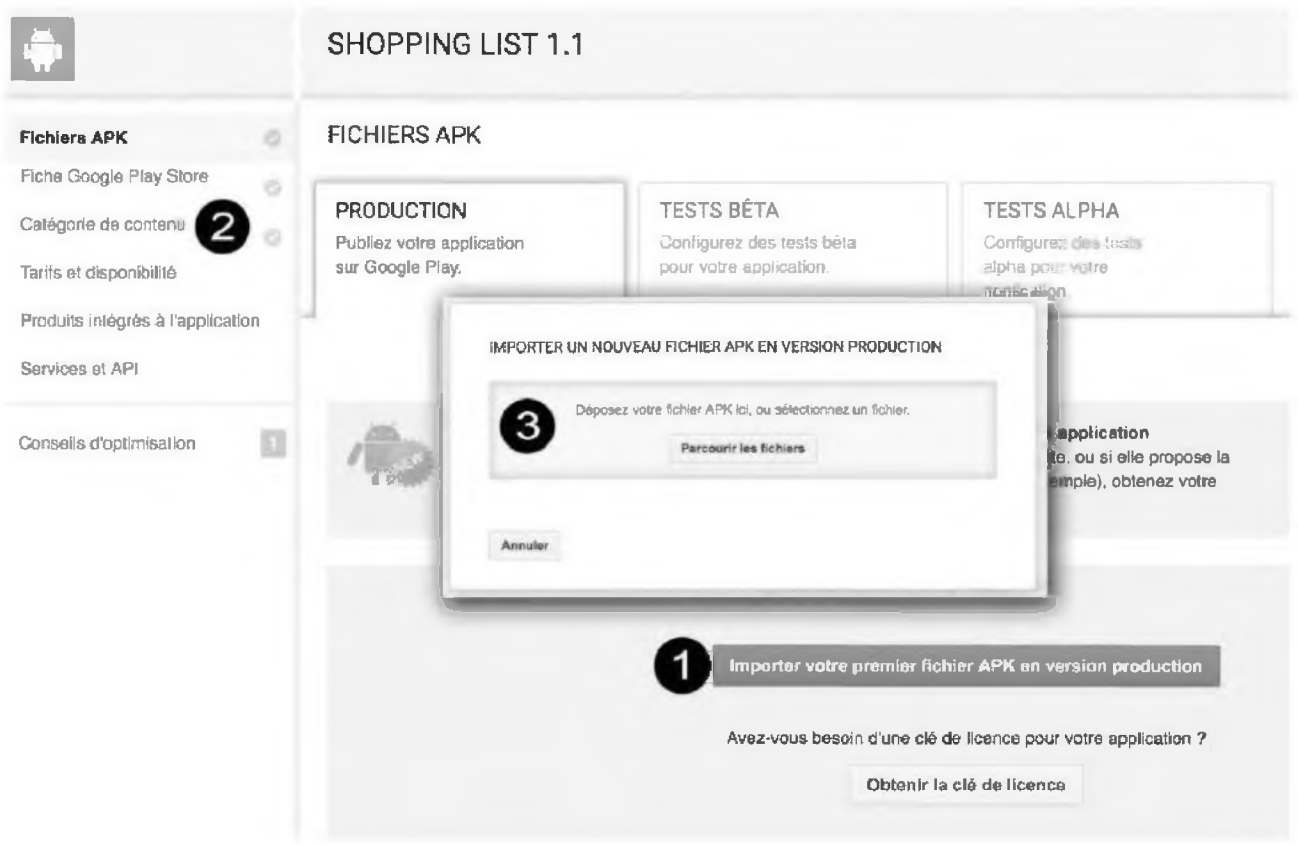


Figure 13-32 *Importer des captures d'écran*

4. Une fois l'application importée, vous devez remplir les fiches Google Play Store et Tarifs et disponibilité en cliquant sur les onglets appropriés (figure 13-32, repère ❷). La fiche Google Play Store vous demande de saisir un grand nombre d'informations (figure 13-33), telles que :
   - le titre ;
   - une description courte ;
   - une description complète ;

Copyright © 2016 Eyrolles.

- plusieurs captures d'écran dont les formats sont explicitement décrits sur le formulaire ;
- une classification décrivant le type de contenu ;
- vos coordoonnées de contact pour les futurs utilisateurs de l'application ;
- accepter les différentes règles de confidentialité liées à la publication et à l'export de contenus.

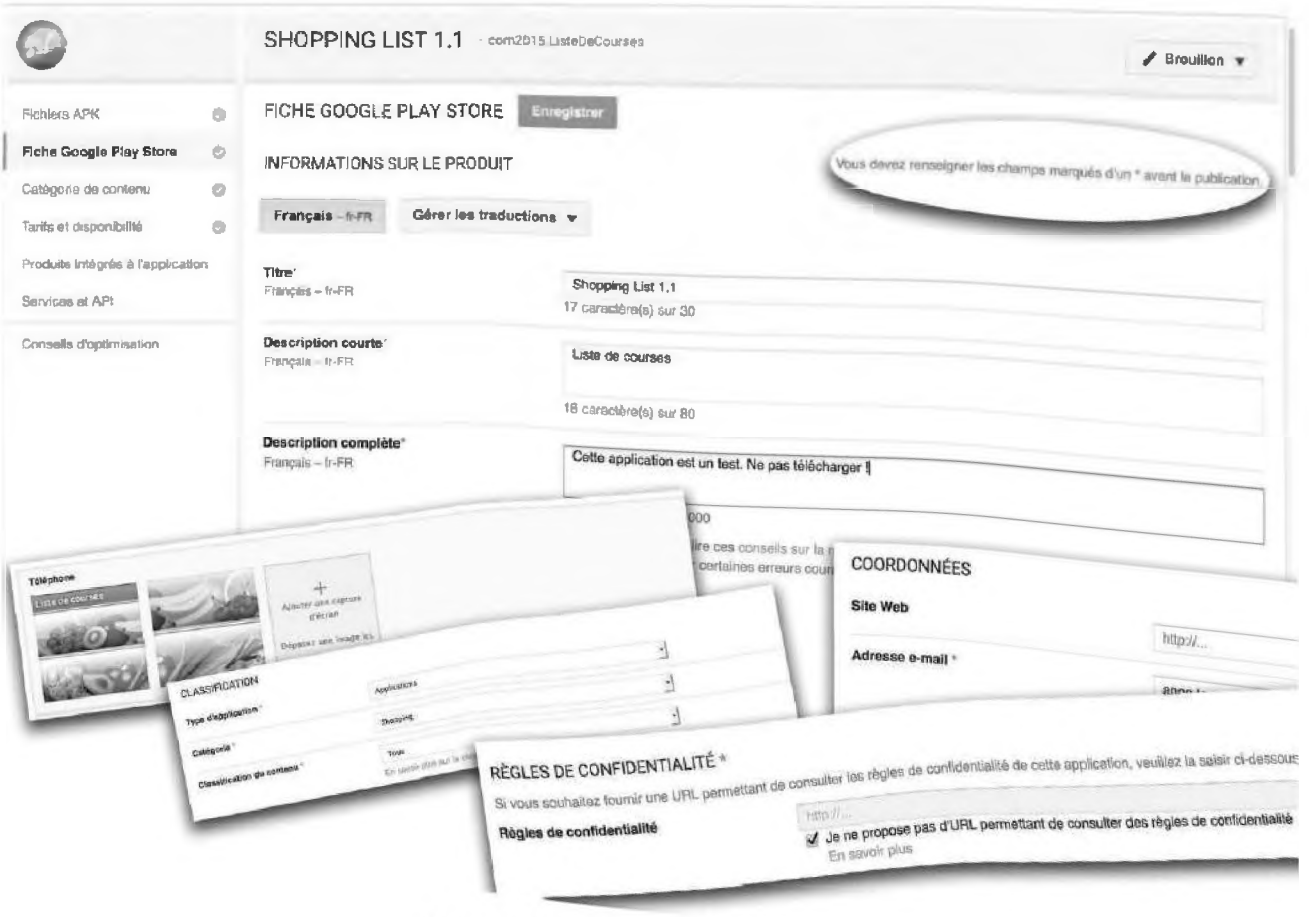


Figure 13-33 *La fiche Google Play Store*

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

5. La fiche Tarifs et disponibilité concerne les options de publication (figure 13-34). Elle permet également de choisir les modalités de vente (gratuit ou payant). Dans le cas d'une application payante, vous devez ouvrir un compte marchand en cliquant sur le lien correspondant. Vous devrez également cocher en bas de page les cases pour signaler votre accord concernant les consignes relatives aux contenus et à l'export aux États-Unis.

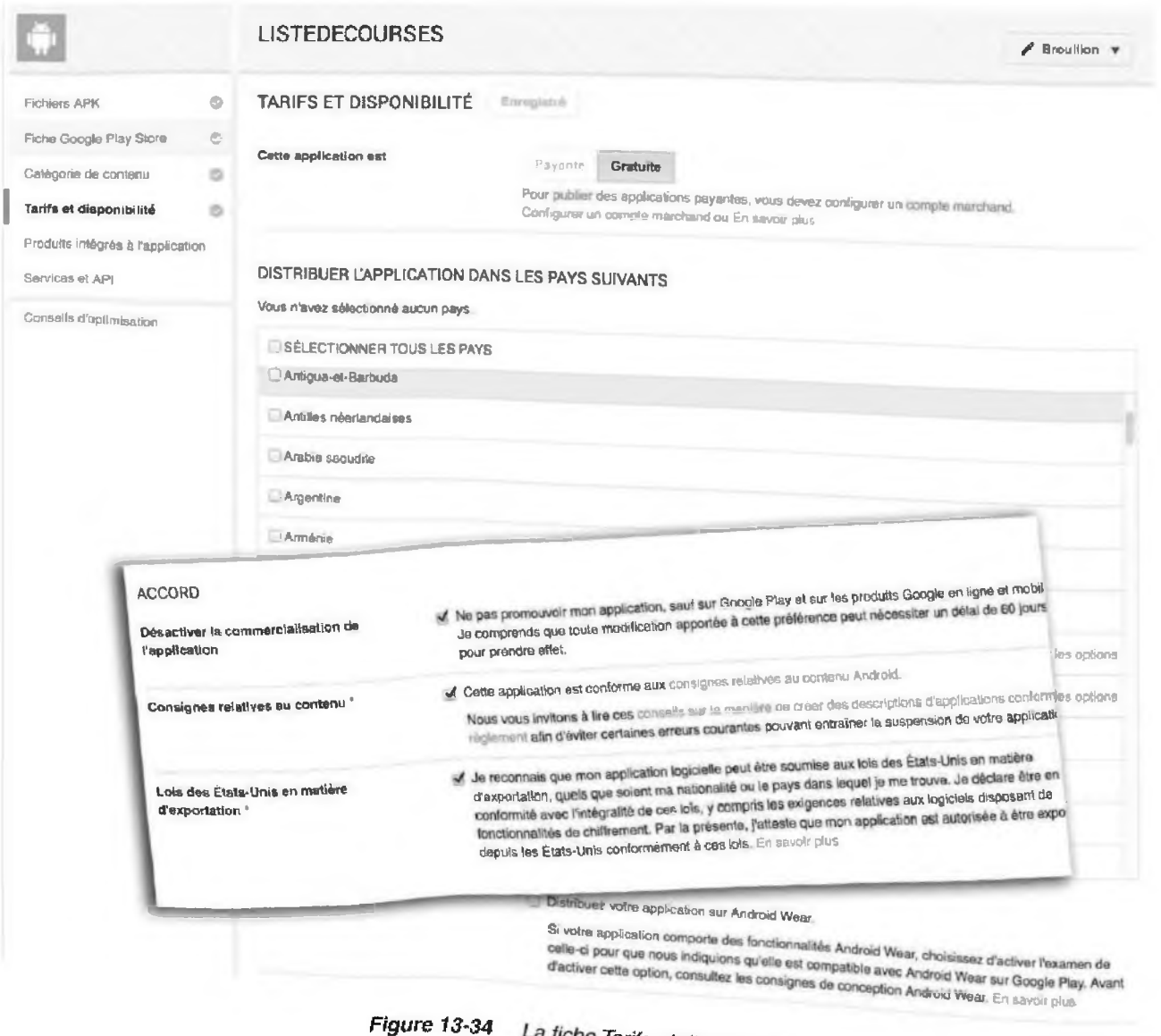


Figure 13-34 *La fiche Tarifs et disponibilité*

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

6. Après avoir rempli l'ensemble des formulaires et lorsque les rubriques Fichiers APK, Fiche Google Play Store et Tarifs et disponiblité sont signalées par une puce verte, vous pouvez cliquez sur le bouton Prête à être publiée – Publier cette application (figure 13-35).

Figure 13-35 *Publier l'application*

La publication de l'application n'est effective que lorsque Google Play affiche la liste de vos applications publiées (figure 13-36).

Figure 13-36 *Votre application est publiée !*

Résumé

Android Studio est une interface de développement adaptée au développement d'applications Android. La création et l'exécution s'effectuent dans le cadre d'un projet structuré selon une arborescence très précise :

- Le répertoire `app` contient les classes Java nécessaires au bon fonctionnement de l'application.
- Le répertoire `res` contient une arborescence de répertoires dans lesquels sont enregistrés tous les fichiers ressources de l'application. Le répertoire `drawable` est utilisé pour stoc-

Copyright © 2016 Eyrolles.

ker les images et le répertoire `layout` comprend tous les fichiers XML descriptifs des composants graphiques nécessaires à la construction de l'interface utilisateur.

Les composants graphiques de l'interface utilisateur sont créés à partir de balises XML spécifiques au composant utilisé. Ainsi, un bouton est défini par les balises suivantes :

```
<Button
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="OK"
    android :id="@+id/okBtn"
/>
```

Pour les attributs `android:layout_width` et `android:layout_height`, la valeur `fill_parent` indique au gestionnaire d'affichage que le bouton a la même taille que son parent, soit le plus souvent la taille de l'écran du téléphone. La valeur `wrap_content` indique, quant à elle, que le bouton s'adapte à son contenu, soit un texte initialisé à OK grâce à l'attribut `text`.

L'attribut `id` est le plus important puisqu'il permet d'identifier chaque composant. La méthode `findViewById()` est ensuite utilisée dans les classes Java pour récupérer le composant défini au sein du fichier XML. Ainsi, l'instruction :

```
Button unBouton = (Button) findViewById(R.id.okBtn);
```

permet d'associer le bouton défini au sein du fichier ressource XML (répertoire `res/layout`) et identifié avec l'attribut `id` égal à `okBtn` avec un objet Java de type `Button`. Le terme `R.id.okBtn` correspond à une variable statique déclarée dans le fichier `R.java` généré directement par Android Studio.

Copyright © 2016 Eyrolles.

Exercices

Comprendre la structure d'un projet Android

L'objectif est ici de construire une application relativement simple afin de mieux appréhender, dans un projet Android, les relations entre les fichiers ressources et le code Java. Cette application reprend l'exemple décrit maintes fois dans cet ouvrage : calculer le périmètre d'un cercle.

L'application se présente sous la forme suivante.

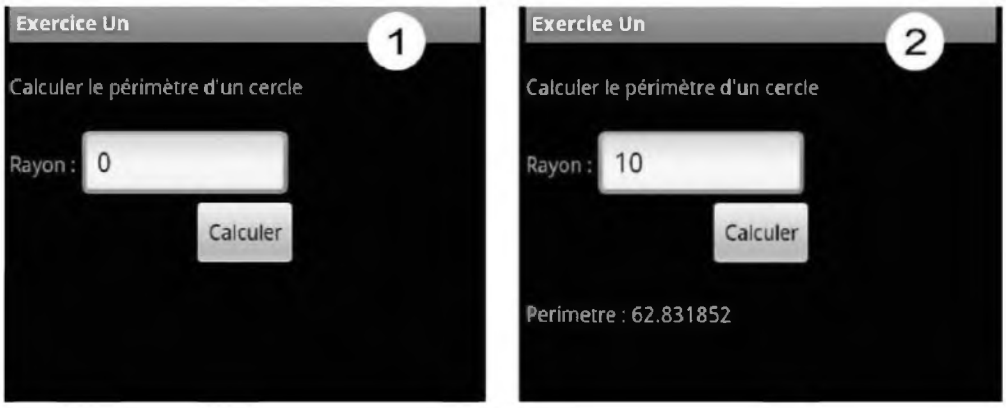


Figure 13-37 *Calculer le périmètre d'un cercle*

L'application `ExerciceUn` propose à l'utilisateur de saisir une valeur pour le rayon. Par défaut, cette valeur est de `0` (figure 13-38, repère ❶). Une fois la valeur saisie, l'utilisateur appuie sur le bouton Calculer. Le résultat s'affiche alors juste en dessous (figure 13-38, repère ❷) avec le message « Perimetre : 62.831852 » si la valeur entrée du rayon vaut `10`.

Mise en place des éléments graphiques (layout/main.xml)

Exercice 13.1

a. À l'aide de l'interface Android Studio, créez un projet `Exercice13_1`, de type `Android`.

b. Dans le fichier `res/values/strings.xml`, modifiez le titre de l'application en `ExerciceUn` au lieu de `Main`.

c. Dans le fichier `res/layout/main.xml`, modifiez le composant `TextView` de manière à afficher le texte « Calculer le périmètre d'un cercle : ». Pour ajouter un espace entre le titre et le contenu du composant `TextView`, utilisez l'attribut `android:layout_marginTop`. Initialisez cet attribut à `20px`.

Copyright © 2016 Eyrolles.

d. Les composants `TextView` et `EditText`, utilisés pour afficher le message « Rayon : » suivi d'un champ de saisie, sont présentés horizontalement. Insérez ces deux composants dans un `LinearLayout` d'orientation horizontale, séparés de 20 pixels par rapport au texte précédent. Initialisez l'attribut `android:layout_marginTop` du `LinearLayout` horizontal à `20px`.

e. Le champ de saisie est défini avec une largeur (`layout_width`) fixée à `140dp`.

Remarque L'unité de mesure dp (densité de pixels indépendants) est une unité abstraite calculée sur la densité physique de l'écran, alors que l'unité px (pixels) correspond aux pixels réels de l'écran. Il est conseillé d'utiliser les dp plutôt que les px pour obtenir des affichages sensiblement équivalents d'un appareil à l'autre.

f. Ajoutez un bouton dont la taille s'adapte à son contenu tant en largeur qu'en hauteur et dont le texte est « Calculer ». Pour centrer le bouton, utilisez l'attribut `android:layout_gravity` en l'initialisant à `center`.

g. Pour afficher le résultat final, créez un dernier composant `TextView` qui s'adapte à son contenu tant en largeur qu'en hauteur. Ce composant ne possède initialement pas de texte. Il est séparé de 20 pixels par rapport au bouton Calculer.

h. Trois composants doivent posséder des identifiants puisqu'ils seront utilisés dans la classe `Main`.
   - `EditText` (id : `valeur`), car vous aurez besoin de récupérer la valeur saisie par l'utilisateur.
   - `Button` (id : `calculer`) pour détecter que l'utilisateur a appuyé dessus.
   - `TextView` (id : `resultat`) pour afficher le résultat.

Définir le comportement du bouton Calculer

Exercice 13.2

a. Le périmètre est calculé lorsque l'utilisateur appuie sur le bouton Calculer. L'activité principale doit donc être en mesure d'écouter des événements de clic. La classe `Main` implémente l'écouteur `OnClickListener`.

b. Dans la méthode `onCreate()`, créez un bouton `calculerBtn` et associez-le au bouton Calculer défini dans le fichier `main.xml`. Pour cela, utiliser la méthode `findViewById()`.

c. Ajoutez un écouteur d'événements sur le bouton `calculerBtn`.

d. Dans la méthode `onClick()`, créez un texte de saisie (`EditText texteIn`) et associez-le au champ de texte `valeur` défini dans le fichier `main.xml`.

e. L'instruction qui permet de récupérer la valeur saisie par l'utilisateur est :

```
String texte = texteIn.getText().toString();
```

Transformez le texte en valeur de type double et calculez le périmètre avec la valeur saisie. Stockez le résultat dans une variable nommée `perimetre` de type double.

Copyright © 2016 Eyrolles.

f. Toujours dans la méthode `onClick()`, créez un texte d'affichage (`TextView resultatText`) et associez-le au texte d'affichage `resultat` défini dans le fichier `main.xml`.

g. Affichez la valeur du périmètre en appliquant la méthode `setText()` à l'objet `resultatText` avec en paramètre l'expression `"Perimetre : " + perimetre`.

La liste des courses – Version 2

Poursuivons ici le développement de l'application `Liste de courses` en écrivant le code du bouton Choisir des légumes.

L'application se présente sous la forme suivante.

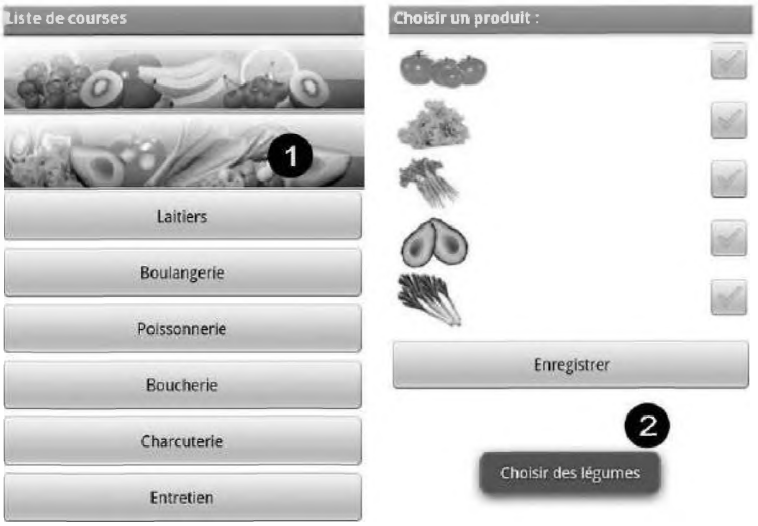


Figure 13-38 *La version 2 de l'application propose une liste de légumes*

Lorsque l'utilisateur appuie sur le bouton Choisir des légumes (voir figure 13-38, repère ❶) :

- Un message éphémère s'affiche, confirmant que la rubrique Choisir des légumes a bien été sélectionnée (voir figure 13-38, repère ❷).
- L'activité `Choisir un produit` apparaît avec une série d'icônes représentant des légumes. À chaque icône est associée une case à cocher.
- Lorsque l'utilisateur appuie sur le bouton Enregistrer, l'activité `Choisir un produit` disparaît et la liste des légumes est mémorisée dans un fichier texte. L'activité principale réapparaît.
- Si l'utilisateur revient sur la rubrique Choisir des légumes, l'activité `Choisir un produit` affiche les cases cochées correspondant aux légumes déjà sélectionnés.

Copyright © 2016 Eyrolles.

Mise en place des ressources (drawable et layout)

Extension Web Vous trouverez tous les codes et fichiers ressources nécessaires à la réalisation de cette application dans le répertoire `Sources/Exercice/Chapitre13/SupportPourRealiser LesExercices/ListeDeCourses`.

Exercice 13.3

a. Ouvrez le projet `ListeDeCoursesAvecLegumes`.

b. Recherchez les fichiers images correspondant aux nouvelles icônes (voir répertoire `images`) et déplacez ces dernières dans le répertoire `drawable` du projet de l'application.

Exercice 13.4

a. Créez un nouveau fichier `legumes.xml` dans le répertoire `layout` du projet de l'application.

b. Insérez les composants graphiques de façon à afficher la liste des icônes et leur case à cocher verticalement. Une icône est placée sur la même ligne que la case à cocher qui lui correspond.

c. Définissez un identifiant pour chaque case à cocher. Il doit porter le nom du légume associé à l'icône.

d. Créez un bouton Enregistrer avec `saveLegume` comme nom d'identifiant.

L'activité RubriqueLegumes

Exercice 13.5

Il s'agit ici de créer la nouvelle activité `RubriqueLegumes` avec tous les composants définis dans le fichier `legumes.xml`. Un message se construit en fonction de l'état des `CheckBox` et s'affiche dans un `Toast`.

a. Créez une nouvelle activité au sein du projet `ListeDeCoursesAvecLegumes`. Nommez-la `RubriqueLegumes`.

b. Créez les différentes `CheckBox`. Grâce à la méthode `findViewById()`, associez-les aux composants définis dans le fichier `legumes.xml`.

c. Créez un bouton `btnSave` et associez-le au composant `saveLegume` défini dans le fichier `legumes.xml`. Ajoutez un écouteur d'événements au bouton `btnSave`.

d. Écrivez la méthode `onClick()` qui vérifie l'état des cases à cocher. Dans le cas où elles sont cochées, créez un message qui s'incrémente du nom du légume et du caractère +.

e. Affichez le message à l'aide d'un `Toast` en évitant d'afficher les caractères +.

f. Ouvrez le fichier `AndroidManifest.xml` et déclarez la nouvelle activité `RubriqueLegumes`.

Copyright © 2016 Eyrolles.

Remarque Attention de ne pas oublier de déclarer la nouvelle activité au sein du projet, sous peine d'obtenir une erreur de compilation.

Exercice 13.6 L'activité `RubriqueLegumes` s'affiche lorsque l'on clique sur le bouton Légumes de l'activité principale.

a. Dans le fichier `Main.java`, repérez les instructions qui créent l'activité `Choisir un fruit`. En vous inspirant de ces lignes de code, créez la nouvelle activité `RubriqueLegumes`.

b. Testez le projet et vérifiez que les composants de l'activité `RubriqueLegumes` et le message s'affichent correctement.

Exercice 13.7 Les choix de l'utilisateur sont mémorisés dans un fichier de type texte.

a. Dans le fichier `RubriqueLegumes.java`, écrivez la méthode `lireListeLegumes()` qui crée un fichier nommé `Legumes.txt`, en `MODE_PRIVATE` pour stocker un message passé en paramètre.

b. Dans la méthode `onClick()`, appelez la méthode `lireListeLegumes()` en passant en paramètre le message construit à l'exercice 13.5.

Exercice 13.8 L'activité `RubriqueLegumes` s'efface lorsque l'utilisateur appuie sur le bouton Enregistrer.

a. Dans le fichier `RubriqueLegumes.java`, écrivez la méthode `fermerLesLegumes()` qui termine et efface l'activité en cours.

b. Dans la méthode `onClick()`, appelez la méthode `fermerLesLegumes()` juste après l'enregistrement des données.

Exercice 13.9 Lorsque l'utilisateur retourne sur la rubrique Légumes, les cases correspondant aux éléments précédemment sélectionnés sont cochées.

a. Dans le fichier `RubriqueLegumes.java`, écrivez la méthode `lireListeLegumes()` qui retourne la chaîne de caractères mémorisée dans le fichier `Legumes.txt`.

b. Écrivez la méthode `majCheckBox()` qui extrait d'un message les mots séparés par les caractères `+`. Pour chaque mot extrait, la méthode modifie l'état des `CheckBox` si ce mot correspond à un nom de légumes.

c. Appelez la méthode `majCheckBox()` dans la méthode `onCreate()`, juste après avoir créé les différentes `CheckBox`.

Copyright © 2016 Eyrolles.

Copyright © 2016 Eyrolles.

Annexe

Guide d'installations

Vous trouverez dans cette annexe un guide pour télécharger tous les outils nécessaires au développement d'applications Java (section « Extension Web »). Toutes les informations concernant l'installation des différents outils de développement d'applications Java (JDK 8, NetBeans) sous Windows, Mac OS X et Linux se trouvent à la section « Installation d'un environnement de développement ».

Vous découvrirez à la section « Utilisation des outils de développement » comment développer des applications Java en « mode commande » ou en utilisant l'environnement de développement NetBeans. L'installation de l'environnement nécessaire au développement d'applications Android via NetBeans y sera aussi détaillée.

Extension Web

À l'adresse *www.annetasso.fr/Java*, vous trouverez tous les liens utiles pour télécharger :

- quatre fichiers :
  - `Corriges.pdf` contient les explications des corrections des exercices et du projet ;
  - `Unicode0000a007F.pdf` donne la liste des 127 premiers caractères de la table Unicode ;
  - `Unicode0080a00FF.pdf` liste les caractères compris entre les indices 128 et 255 de la table Unicode ;
  - `Sources.zip` contient tous les codes sources des exemples, des exercices corrigés et du projet ;
- les environnements :
  - Java ;
  - NetBeans ;
  - Android Studio.

Le fichier corriges.pdf

Ce fichier, au format PDF (à lire avec le logiciel Acrobat Reader), fournit, pour chaque chapitre du livre, le corrigé **commenté** et **expliqué** des exercices et du projet.

La correction se présente sous la forme de codes sources commentés et d'explications détaillées sur des points techniques demandant davantage de précisions. Ainsi, par exemple, la correction de l'exercice 3.5 du chapitre 3, « Faire des choix », est exposée comme suit.

Copyright © 2016 Eyrolles.

Énoncé

Exercice 3.5 En utilisant la structure `switch`, écrire un programme qui simule une machine à calculer dont les opérations sont l'addition (+), la soustraction (–), la multiplication (*) et la division (/).

a. En cours d'exécution, le programme demande à l'utilisateur d'entrer deux valeurs numériques, puis le caractère correspondant à l'opération à effectuer. Suivant le caractère saisi (+, –, * ou /), le programme affiche l'opération effectuée ainsi que le résultat.

L'exécution du programme peut, par exemple, avoir l'allure suivante (les valeurs grisées sont celles saisies par l'utilisateur) :

```
Entrez la première valeur : 2
Entrez la seconde valeur : 3
Type de l'opération (+, -, *, /) : *
Cette opération a pour résultat : 2 * 3 = 6
```

b. Après avoir écrit et exécuté le programme avec différentes valeurs, saisissez dans cet ordre les valeurs suivantes : `2`, `0`, puis le caractère `/`. Que se passe-t-il ? Pourquoi ?

c. Modifiez le programme de façon à ne plus rencontrer cette situation en cours d'exécution.

Corrigé

a. Le code source complet

```
import java.util.*;
public class Calculette {
 public static void main( String [] argument) {
  int a, b;
  char opérateur;
  double calcul = 0;
  Scanner lectureClavier = new Scanner(System.in);
  // Lire et stocker la première valeur dans a
  System.out.print("Entrer la première valeur : ");
  a = lectureClavier.nextInt();
  // Lire et stocker la première valeur dans b
  System.out.print("Entrer la seconde valeur : ");
  b = lectureClavier.nextInt();
  // Lire et stocker le signe de l'opération dans l'opérateur
  System.out.print("Type de l'opération : (+, -, *, /) : ");
  opérateur = lectureClavier.next().charAt(0);
  // suivant le signe de l'opération
  switch (opérateur ) {
   // Si c'est le caractère +, faire une addition
   case '+' : calcul = a + b;
              break;
   // Si c'est le caractère -, faire une soustraction
```

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

```
    case '-' : calcul = a - b;
               break;
    // Si c'est le caractère n /, faire une division
    case '/' : calcul = a / b;
               break;
    // Si c'est le caractère *, faire une multiplication
    case '*' : calcul = a * b ;
               break;
    }
    // Afficher le résultat
    System.out.print("Cette opération a pour résultat : ");
    System.out.println(a+" "+ opérateur +" "+ b + " = " + calcul);
  }
}
```

b. Exécution du programme avec le jeu de valeurs 2, 0 et /

```
Entrer la première valeur : 2
Entrer la seconde valeur : 0
Type de l'opération : (+, -, *, /) : /
java.lang.ArithmeticException: / by zero
at Calculette.main(Calculette.java:22)
```

L'interpréteur détecte une exception de type arithmétique. Il s'agit de la division par zéro.

c. Correction du code source

L'erreur provient de la division. Il suffit de vérifier que la valeur de `b` est non nulle pour l'étiquette `'/'` de la structure `switch`. Examinons la correction :

```
import java.util.*;
public class Calculette{
 public static void main( String [] argument) {
  int a, b;
  char opérateur;
  double calcul = 0;
  // Définir et initialiser un booléen à true
  boolean OK = true;
  Scanner lectureClavier = new Scanner(System.in);
  System.out.print("Entrer la première valeur : ");
  a = lectureClavier.nextInt();
  System.out.print("Entrer la seconde valeur : ");
  b = lectureClavier.nextInt();
  System.out.print("Type de l'opération : (+, -, *, /) : ");
  opérateur = lectureClavier.next().charAt(0);
  switch (opérateur ) {
```

Copyright © 2016 Eyrolles.

```
    case '+' : calcul = a + b;
               break;
    case '-' : calcul = a - b;
               break;
    // Si c'est le caractère /, tester la valeur de b
    case '/' : if ( b != 0) calcul = a / b;
               else
               // Si b vaut 0, mettre le booléen OK à false
               OK = false;
               break;
    case '*' : calcul = a * b;
               break;
    default : OK = false;
     }
    // Si OK vaut true, afficher le résultat
    if (OK) {
      System.out.print("Cette opération a pour résultat : ");
      System.out.println(a+" "+opérateur+ " "+ b + " =  " + calcul);
    }
    // Si OK vaut false, afficher un message d'erreur
    else System.out.println("Opération non conforme !");
  }
}
```

À l'étiquette `'/'`, le programme vérifie si le contenu de la variable `b` est non nul. Si tel est le cas, il exécute normalement l'instruction réalisant la division. En revanche, si le contenu de la variable `b` est nul, la division n'est pas effectuée mais la valeur `false` est affectée à la variable `OK` de type booléen (initialisée par défaut à `true` lors de la déclaration de la variable).

Ensuite, pour afficher le résultat du calcul, le programme vérifie la valeur de la variable `OK`. Si elle vaut `true`, cela signifie que l'opération a été effectuée sans rencontrer de difficulté particulière ; sinon, cela signifie qu'aucune opération n'a pu être réalisée. Le programme signale alors par un message que l'opération est non conforme.

Remarquez que la valeur `false` est aussi affectée à la variable `OK` pour l'étiquette `default`. Ainsi, si l'utilisateur saisi un caractère autre que `+`, `-`, `/` ou `*`, le programme n'exécute aucun calcul et signale par un message que l'opération est non conforme.

Dans le jargon informatique, on dit que la variable `OK` est un drapeau (*flag* en anglais). En effet, il change d'état (de valeur) en fonction des instructions exécutées. Ce terme « drapeau » fait allusion au système de fonctionnement des boîtes aux lettres américaines munies d'un drapeau rouge. Lorsque le facteur dépose du courrier, le drapeau est relevé ; il l'abaisse pour indiquer la présence de courrier. Lorsque le destinataire récupère son courrier, il relève le drapeau, indiquant que la boîte est désormais vide. Ainsi, la position (état) du drapeau indique la présence (drapeau abaissé) ou non (drapeau levé) de courrier dans la boîte aux lettres.

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

L'archive Sources.zip

Après avoir téléchargé et décompressé le fichier `Sources.zip`, vous obtenez un dossier `Sources` contenant trois sous-dossiers : `Exemples`, `Exercices` et `Projet`. Ceux-ci contiennent respectivement douze répertoires, un pour chacun des chapitres : `Introduction`, `Chapitre1`, `Chapitre2`, `Chapitre3`... `Chapitre12`.

Chacun de ces répertoires contient les fichiers sources des programmes :

- correspondant aux exemples. Ainsi, pour retrouver les programmes donnés en exemples au chapitre 1, rendez-vous dans le répertoire `Sources/Exemples/Chapitre1`.
- correspondant aux exercices corrigés. Ainsi, pour retrouver le programme de l'exercice 2 du chapitre 4, allez dans le répertoire `Sources/Exercices/Chapitre4`.
- correspondant au projet. Ainsi, pour retrouver le corrigé du projet du chapitre 5, ouvrez le répertoire `Sources/Projet/Chapitre5`.

Le lien Java

En cliquant sur le lien Java, vous pourrez télécharger les programmes d'installation du JDK Java pour Mac, Linux et Windows.

Pour installer le JDK (*Java Development Kit*), reportez-vous à la section qui vous intéresse en fonction de votre système d'exploitation : « Installation de Java SE Development Kit sous Windows », « Installation de Java SE Development Kit sous Linux » ou encore « Installation de Java SE Development Kit sous Mac OS X » ci-après.

Le lien NetBeans

En cliquant sur le lien NetBeans, vous pourrez télécharger les programmes d'installation de l'environnement de développement NetBeans pour Mac, Linux et Windows.

Pour installer NetBeans, reportez-vous à la section « Installation de NetBeans sous Windows », « Installation de NetBeans sous Linux » ou encore « Installation de NetBeans sous Mac » en fonction de votre système d'exploitation.

Le lien Android Studio

En cliquant sur le lien Android Studio, vous pourrez télécharger les programmes d'installation de l'environnement de développement Android Studio pour Mac, Linux et Windows.

Pour installer Android Studio, reportez-vous à la section « Développer des applications Android avec Android Studio ».

Copyright © 2016 Eyrolles.

Installation d'un environnement de développement

Java SE (*Java Standard Edition*) est la plate-forme de base fournie par Sun pour l'exécution des applications Java. Java SE intègre par défaut le JDK, environnement de développement indispensable pour compiler les classes de vos applications.

Pour exécuter l'ensemble des exemples présentés dans cet ouvrage et réaliser les exercices proposés à la fin de chaque chapitre, vous devez créer votre propre environnement de travail.

Pour cela, vous devez installer :

- le compilateur Java nécessaire à l'exécution des programmes donnés en exemples ou pour réaliser les exercices ;
- l'environnement de développement NetBeans ;
- les outils permettant le développement d'applications Android.

Dans les sections suivantes, vous trouverez toutes les informations nécessaires à l'installation de ces éléments sous Windows 2000, NT, XP, Vista, Windows 7 & 8, Mac OS X et Linux.

Installation de Java SE Development Kit sous Windows

Dans la section suivante, nous détaillons la procédure d'installation de la dernière version de Java sur Windows 2000, XP, NT, Vista et 7.

La procédure d'installation

Une fois le fichier d'installation `jdk-8u66-windows-x64-p.exe` téléchargé depuis l'extension Web, déplacez-le sur le Bureau de votre ordinateur.

Voici la marche à suivre pour installer l'environnement Java.

1. Double-cliquez sur le fichier `jdk-8u66-windows-x64-p.exe`. Une fenêtre de préparation de l'installation apparaît. Laissez l'ordinateur charger l'assistant d'installation en mémoire, puis cliquez sur le bouton Next.

Figure A-1

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

2. Vous pouvez changer le répertoire d'installation de l'environnement en cliquant sur le bouton Browse.

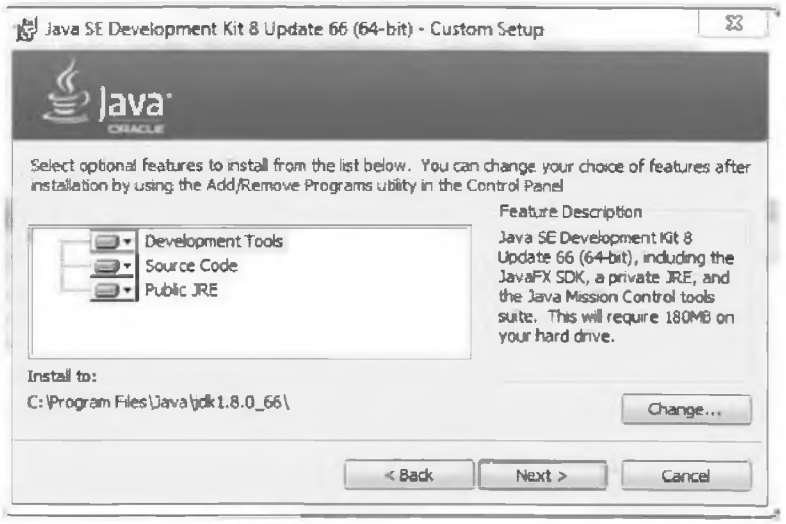


Figure A-2

Attention Notez le répertoire d'installation de Java SE. Le chemin correspondant est utilisé pour définir la variable d'environnement `PATH` (voir section « Les variables d'environnement » ci-après). Par défaut, Java SE 1.8.0_66 s'installe dans le répertoire `C:\Program Files\Java\jdk1.8.0_66`.

3. Cliquez sur le bouton Next pour lancer l'installation.
4. Une barre de progression vous indique l'état d'avancement de l'installation.

Figure A-3

Copyright © 2016 Eyrolles.

5. Une fois l'installation terminée, cliquez sur le bouton Close. Il n'est pas nécessaire de redémarrer votre ordinateur.

Figure A-4

6. Supprimez le fichier `jdk-8u66-windows-x64-p.exe`.

La variable d'environnement PATH

Pour compiler et exécuter un programme Java, il est nécessaire d'utiliser la commande de compilation `javac` fournie par Java SE. Elle est définie dans le répertoire par défaut, `C:\Program Files\Java\jdk1.8.0_66\bin` ou bien dans le répertoire `C:\leRepertoire\deVotre\choix\jdk1.8.0_66\bin`, déterminé au moment de l'installation de Java SE.

Pour utiliser la commande de compilation `javac` depuis un autre répertoire, il est nécessaire « d'expliquer » à l'ordinateur par quel chemin y accéder via la variable d'environnement `PATH`.

Sous Windows 2000, XP et NT

La mise en place de la variable d'environnement `PATH` s'effectue de la façon suivante :

1. Dans le menu Démarrer (puis l'onglet Paramètres sous Windows 2000), sélectionnez Panneau de configuration.

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

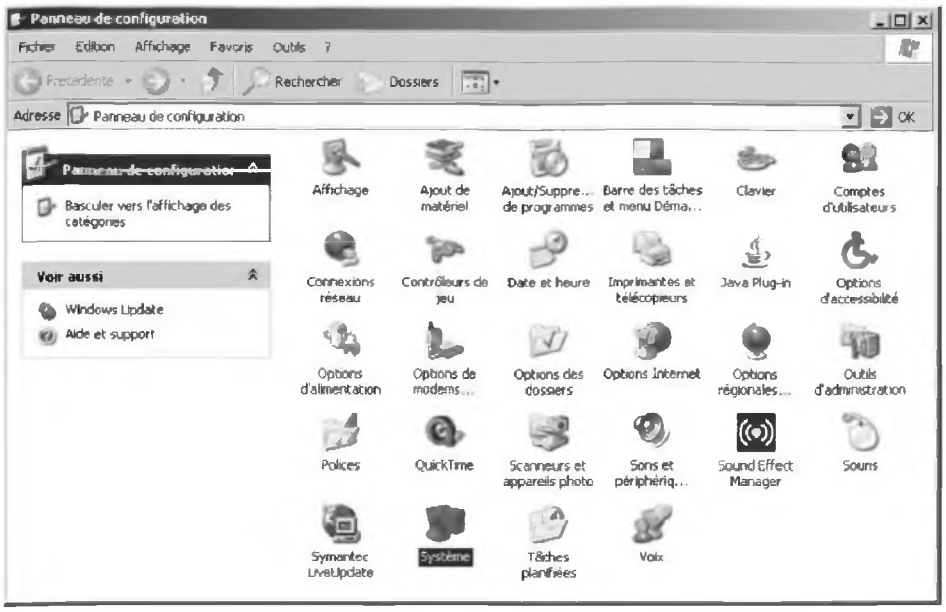


Figure A-5

2. Double-cliquez sur Système et sélectionnez l'onglet Avancé.

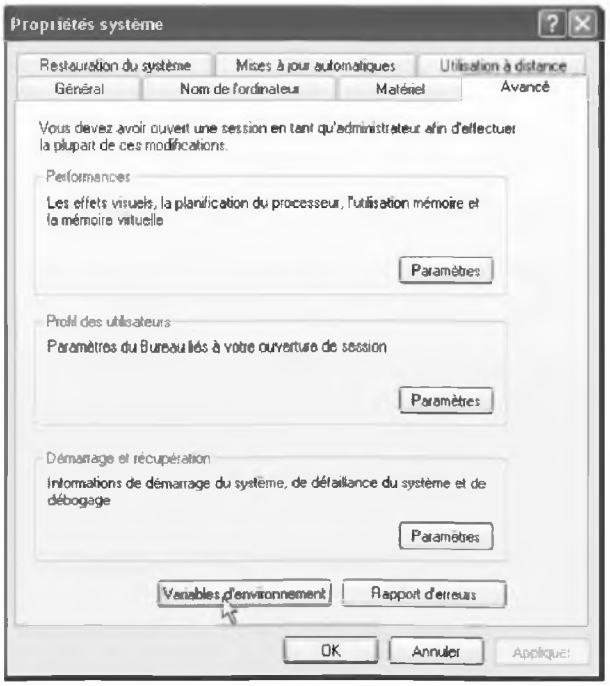


Figure A-6

Copyright © 2016 Eyrolles.

3. Cliquez sur le bouton Variables d'environnement.

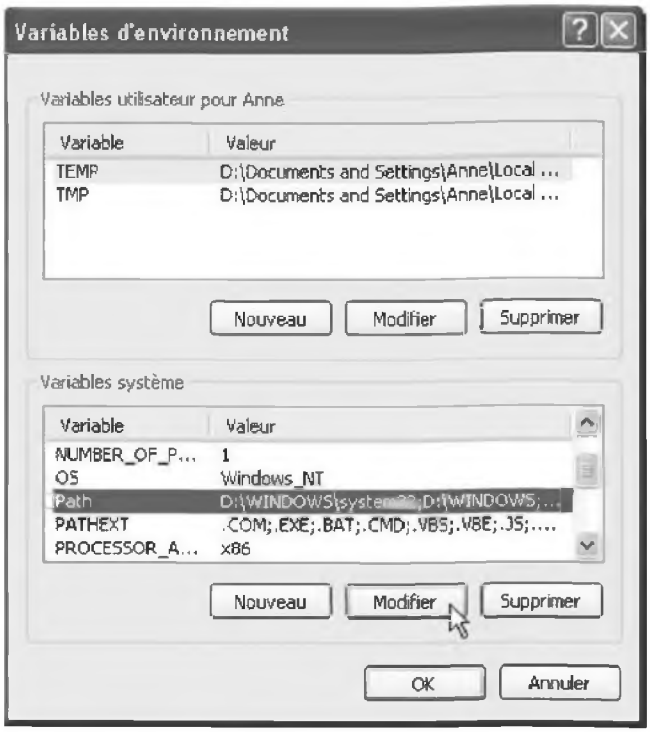


Figure A-7

4. Dans la rubrique Variables système, sélectionnez la variable `PATH`. Cliquez sur le bouton Modifier, ajoutez la ligne `C:\Program Files\Java\jdk1.8.0_66\bin;` ou `C:\le\repertoire\de\votre\choix\jdk1.8.0_66\bin;` dans le champ Valeur de la variable.

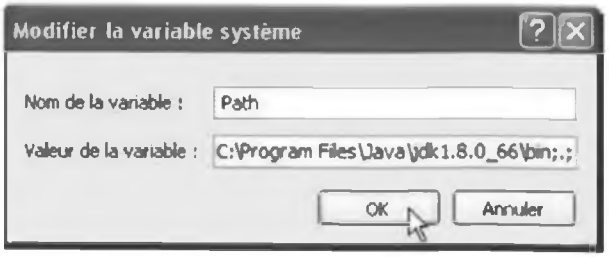


Figure A-8

5. Cliquez sur les boutons OK des différentes fenêtres pour les fermer et valider la nouvelle configuration.

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

Remarque Lorsque vous modifiez la variable `PATH` du système, veillez à bien ajouter la nouvelle ligne sans supprimer le contenu précédent. Pour cela, saisissez le signe ; à la fin de la ligne existante et ajoutez à la suite `C:\Program Files\Java\jdk1.8.0_66\bin;`.

Attention, le fait de supprimer le contenu précédent peut nuire au bon fonctionnement des autres applications installées sur votre ordinateur.

Sous Windows Vista et 7

La mise en place de la variable `PATH` s'effectue de la façon suivante :

1. Dans le menu Démarrer, sélectionnez Panneau de configuration.

Figure A-9

Remarque Si la fenêtre Panneau de configuration ne s'affiche pas comme celle présentée en figure A-9, cliquez sur le lien Affichage classique situé sur la partie gauche de la fenêtre.

Copyright © 2016 Eyrolles.

2. Double-cliquez sur Système, puis cliquez sur le lien Paramètres système avancés situé à gauche.

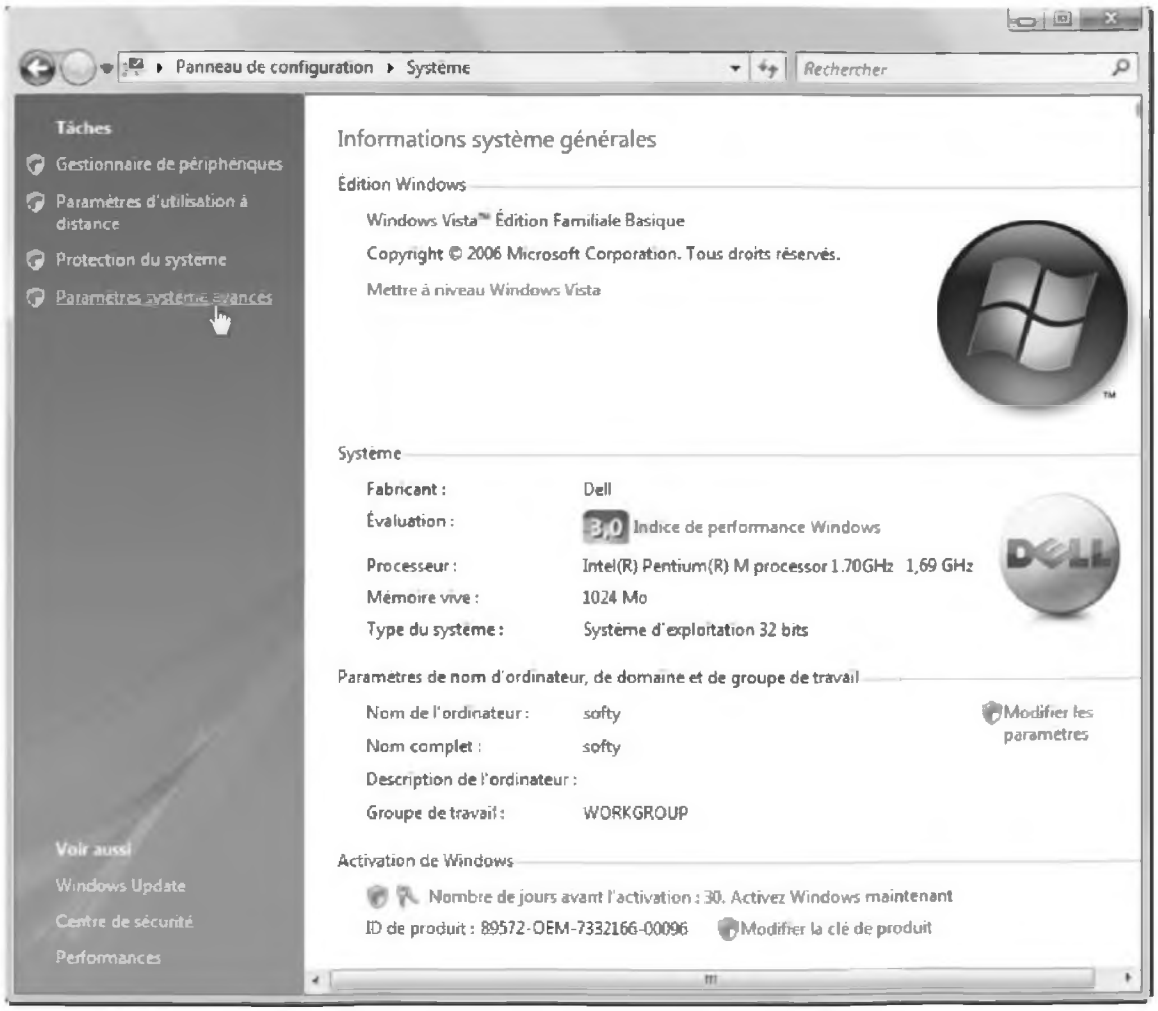


Figure A-10

Copyright © 2016 Eyrolles.

3. Cliquez sur le bouton Variables d'environnement.

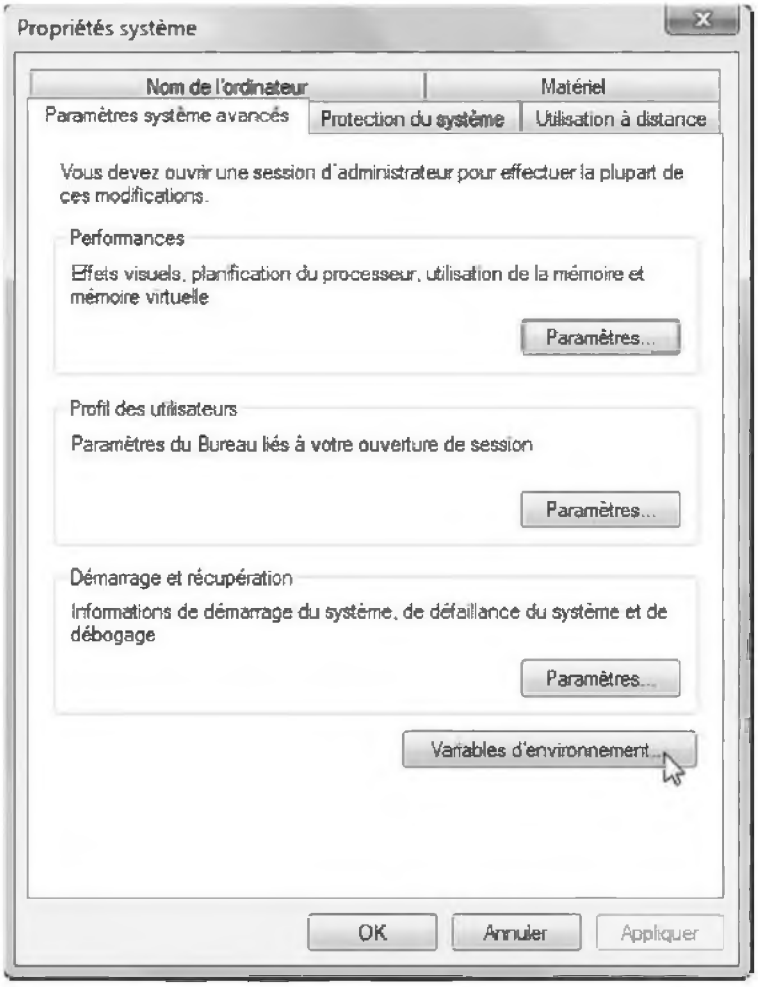


Figure A-11

Copyright © 2016 Eyrolles.

4. Dans la rubrique Variables système, sélectionnez la variable `PATH` et cliquez sur le bouton Modifier.

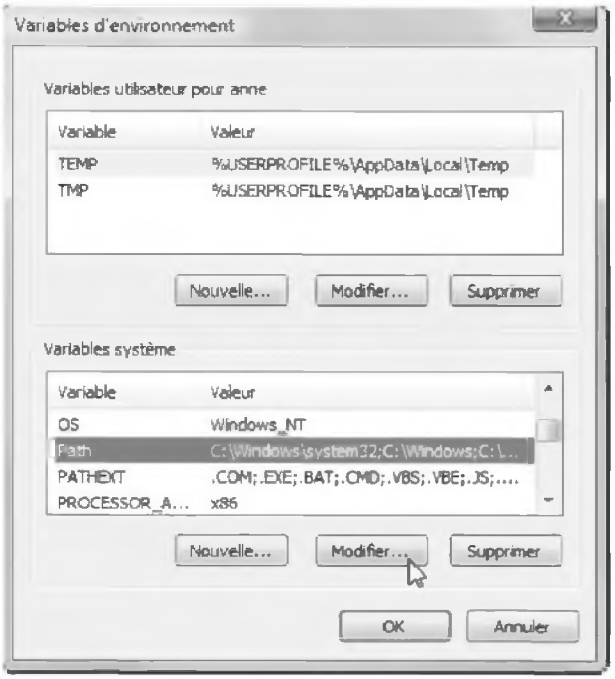


Figure A-12

5. Ajoutez la ligne `C:\Program Files\Java\jdk1.8.0_66\bin;` ou `C:\le\repertoire\de\votre\choix\jdk1.8.0_66\bin;` dans le champ Valeur de la variable.

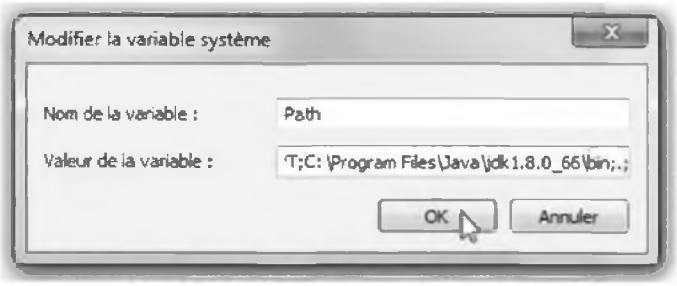


Figure A-13

6. Cliquez sur les boutons OK des différentes fenêtres pour les fermer et valider la nouvelle configuration.

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

Remarque

Lorsque vous modifiez la variable `PATH` du système, veillez à bien ajouter la nouvelle ligne sans supprimer le contenu précédent. Pour cela, insérez le signe ; à la fin de la ligne existante et ajoutez à la suite `C:\Program Files\Java\jdk1.8.0_25\bin;`.

Attention, le fait de supprimer le contenu précédent peut nuire au bon fonctionnement des autres applications installées sur votre ordinateur.

La variable CLASSPATH

La variable `CLASSPATH` est utilisée au moment de la compilation d'applications composées de plusieurs fichiers Java. La procédure de mise en place de cette variable est identique à celle de `PATH` décrite précédemment. Dans la fenêtre Modifier la variable système, entrez comme nom de variable `CLASSPATH` et comme valeur `C:\leRepertoire\ouSeTrouvent\lesDifferentes\classes\aCompiler;`.

Pour écrire, compiler et exécuter votre premier programme Java, reportez-vous à la section « Utilisation des outils de développement » plus loin dans ce chapitre.

Installation de Java SE Development Kit 8 sous Mac OS X

La version de Java à installer dépend du système d'exploitation de votre Mac. Ainsi, Java SE 8 n'est disponible que pour les versions Mac OS X 10.7 et ultérieures.

Rendez-vous à l'adresse *www.annetasso.fr/Java* et téléchargez le fichier d'installation de Java en cliquant sur le lien correspondant à votre système d'exploitation sous Mac :

- pour Mac OS X 10.7 et supérieur, vous obtenez le fichier `jdk-8u66-macosx-x64.dmg` ;
- pour Mac OS X 10.6, vous obtenez le fichier `JavaForMacOSX10.6Update16.dmg` ;
- pour Mac OS X 10.5, vous obtenez le fichier `JavaForMacOSX10.5Update10.dmg` ;
- pour les versions antérieures à Mac OS X 10.5 et supérieures à la version 10.4.10, téléchargez le fichier d'installation `JavaForMacOSX10.4Release9.dmg`, qui installe la version JSE 5.0.

L'installation

Voici la marche à suivre pour installer l'environnement Java (Java SE 8).

1. Selon la version de votre système d'exploitation, copiez le fichier d'extension `.dmg` (ici, `jdk-8u66-macosx-x64.dmg`) sur le Bureau de l'ordinateur, puis double-cliquez dessus.

Copyright © 2016 Eyrolles.

2. Une fenêtre contenant l'icône `JDK 8 Update 25.pkg` apparaît : double-cliquez dessus.

Figure A-14

3. La fenêtre d'installation de Java s'affiche. Cliquez sur le bouton Continuer.

Figure A-15

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

4. Une fenêtre indiquant le mode d'installation et l'espace requis par Java apparaît. Cliquez sur le bouton Installer pour lancer l'installation.

Figure A-16

5. Saisissez le mot de passe associé à votre compte utilisateur Mac pour autoriser l'installation de Java puis cliquez sur Installer le logiciel.

Figure A-17

Copyright © 2016 Eyrolles.

6. L'installation démarre et son état d'avancement est indiqué par une barre de progression.

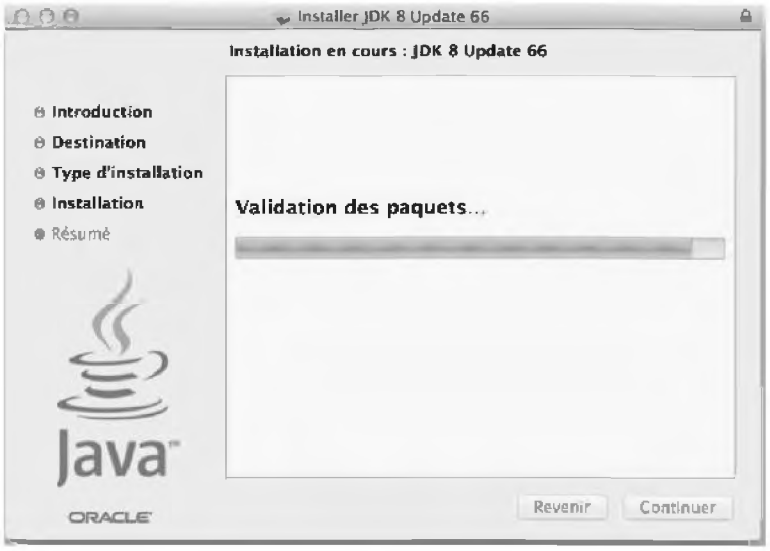


Figure A-18

7. Une fois l'installation terminée, cliquez sur le bouton Fermer. Il n'est pas nécessaire de redémarrer votre ordinateur.

Figure A-19

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

8. Éjectez le paquet `JDK 8 Update 25.pkg` et supprimez le fichier `jdk-8u66-macosx-x64.dmg`.

Pour écrire, compiler et exécuter votre premier programme Java, reportez-vous à la section « Utilisation des outils de développement » plus loin dans ce chapitre.

Installation de Java SE Develoment Kit 8 sous Linux

Afin d'éviter tout problème de droit, il est conseillé d'effectuer l'installation de Java SE sous l'utilisateur `root` en tapant les commandes suivantes :

```
sudo su - root
Password : *****
```

Remarque Sous Ubuntu, vous trouverez directement le paquet `sun-java8-jdk` par l'intermédiaire du gestionnaire de paquets.

L'installation

Une fois le fichier d'installation `jdk-8u66-linux-x64.tar.gz` téléchargé depuis l'extension Web, déplacez-le sur le Bureau de l'ordinateur.

Pour installer l'environnement Java, la marche à suivre est la suivante :

1. Déplacez-vous dans le répertoire où vous souhaitez installer Java SE, par exemple à l'aide de la commande :

   ```
   cd /usr/local/
   ```

2. Décompressez le fichier d'installation `jdk-8u66-linux-x64.tar.gz` grâce à la commande :

   ```
   tar zxvf jdk-8u66-linux-x64.tar.gz
   ```

3. Les fichiers du kit de développement Java s'installent dans le répertoire `/usr/local`.
4. L'installation se termine par la création du répertoire `jdk1.8.0` et de ses sous-répertoires sur le disque dur.
5. Pour simplifier l'appel aux commandes Java, créez un lien symbolique vers le répertoire `jdk1.8.0`, en utilisant la commande :

   ```
   ln -s jdk1.8.0 java
   ```

   L'environnement d'exécution et de développement de Java est maintenant installé sur le disque.
6. Supprimez enfin le fichier d'installation `jdk-8u66-linux-x64.tar.gz` et fermez le compte `root`.

Copyright © 2016 Eyrolles.

Les variables d'environnement

La variable PATH

Pour compiler et exécuter un programme Java, il est nécessaire d'utiliser la commande de compilation `javac` fournie par J2SE. Elle est définie dans le répertoire par défaut `/usr/local/jdk1.8.0/bin`.

Pour utiliser la commande de compilation `javac` depuis un autre répertoire, il est nécessaire « d'expliquer » à l'ordinateur par quel chemin y accéder via la variable d'environnement `PATH`.

La définition de cette variable s'effectue en modifiant le fichier `/home/votreCompte/.bashrc` de votre compte utilisateur, en y ajoutant la ligne suivante :

```
PATH=$PATH :/usr/local/java/bin; export PATH
```

Exécutez votre profil `.bashrc` en tapant la commande :

```
. .bashrc
```

La variable CLASSPATH

La variable `CLASSPATH` est utilisée au moment de la compilation d'applications composées de plusieurs fichiers Java. La définition de cette variable s'effectue en modifiant le fichier `/home/votreCompte/.bashrc` de votre compte utilisateur, en y ajoutant les lignes suivantes :

```
CLASSPATH =\leRepertoire\ouSeTrouvent\lesDifferentes\classes\
aCompiler\.
export CLASSPATH
```

Exécutez votre profil `.bashrc` en tapant la commande :

```
. .bashrc
```

Installation de NetBeans sous Windows 2000, NT, XP, Vista et 7

NetBeans est un environnement de développement intégré (IDE pour *Integrated Development Environment*) développé par Sun. La plate-forme NetBeans propose un espace de travail mêlant les fonctions d'édition et de compilation de programmes écrits en Java.

L'installation

Pour installer l'environnement NetBeans, la démarche est la suivante :

1. Une fois le fichier `netbeans-8.1-javase-windows.exe` téléchargé depuis l'extension Web, déplacez-le sur le Bureau de votre ordinateur.
2. Lancez l'installation de l'environnement NetBeans en double-cliquant sur l'icône du fichier `netbeans-8.1-javase-windows.exe`.

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

3. La fenêtre de préparation de l'installation apparaît. Laissez l'ordinateur charger l'assistant d'installation en mémoire.
4. Lorsque l'assistant d'installation est chargé et prêt, cliquez sur le bouton Next.

Figure A-20

5. Acceptez les termes de la licence, puis ceux de la licence JUnit avant de poursuivre l'installation. Pour cela, cochez l'option I accept the terms... des deux fenêtres successives. Cliquez ensuite sur le bouton Next.

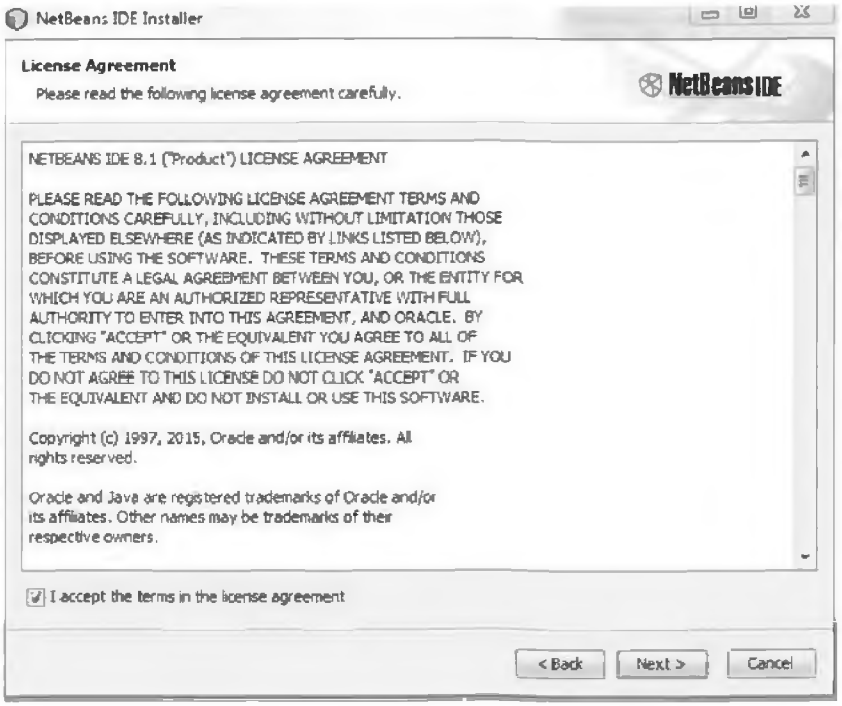


Figure A-21

Copyright © 2016 Eyrolles.

6. Vous pouvez modifier le répertoire d'installation de l'environnement en cliquant sur le bouton Browse..., puis sur Next.

Figure A-22

7. Cliquez sur le bouton Install.

Figure A-23

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

8. L'installation démarre et son état d'avancement est indiqué par une barre de progression.

Figure A-24

9. Une fois l'installation terminée, cliquez sur le bouton Finish et cochez ou décochez les options proposées. Il n'est pas nécessaire de redémarrer votre ordinateur.

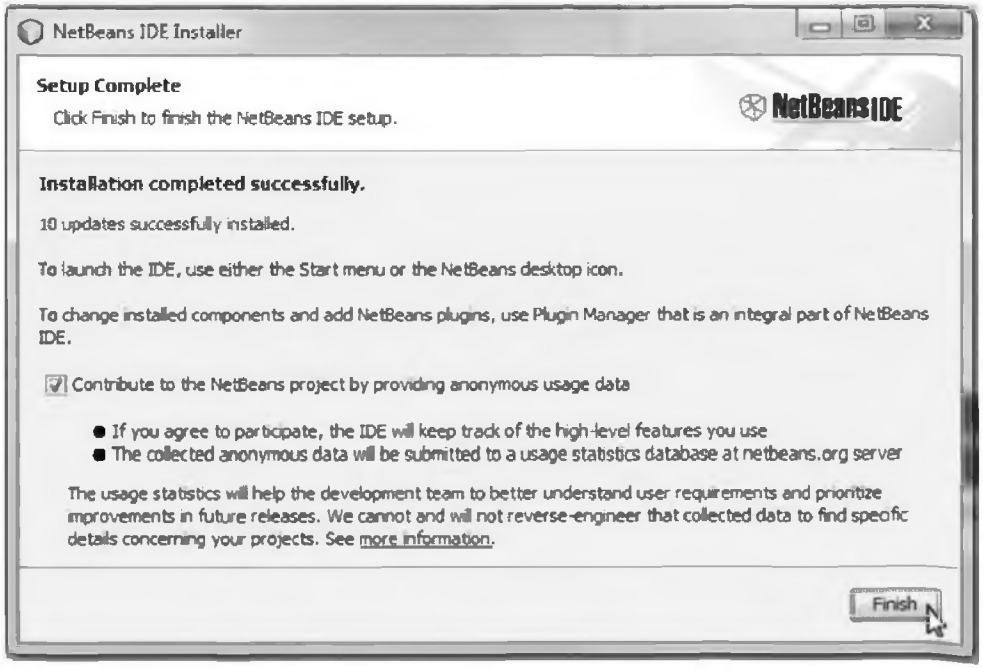


Figure A-25

10. Pour finir, supprimez le fichier d'installation `netbeans-8.1-ml-javase-windows.exe`.

Copyright © 2016 Eyrolles.

Lancement de l'environnement NetBeans

Pour lancer NetBeans, il suffit de double-cliquer sur l'icône `NetBeans` située sur votre Bureau. L'environnement NetBeans s'ouvre alors et la fenêtre de présentation de l'application apparaît.

Figure A-26

Une fois lancé, NetBeans affiche à l'écran un ensemble de fenêtres vides, ainsi qu'une page d'accueil.

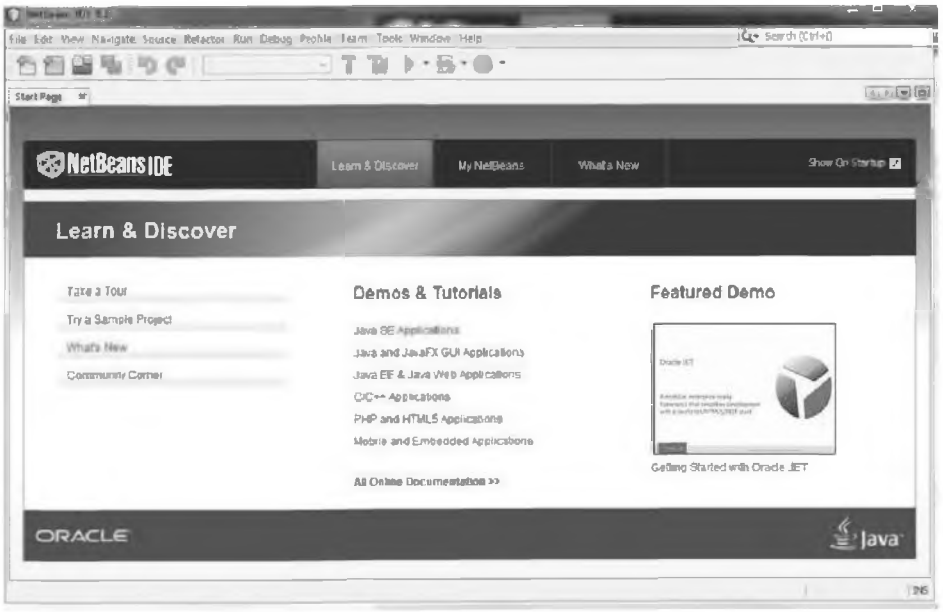


Figure A-27

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

La description et l'utilisation de l'environnement de développement d'applications Java seront décrites à la section suivante « Utilisation des outils de développement – Développer avec NetBeans ».

Installation de NetBeans sous Mac OS X 10.7 et supérieur

Une fois le fichier d'installation `netbeans-8.1-javase-macosx.dmg` téléchargé depuis l'extension Web, déplacez-le sur le Bureau de votre ordinateur.

L'installation

Pour installer l'environnement NetBeans, la marche à suivre est la suivante :

1. Double-cliquez sur le fichier `netbeans-8.1-javase-macosx.dmg`.
2. Une fenêtre contenant une icône intitulée `NetBeans 8.1.pkg` apparaît : double-cliquez dessus.

Figure A-28

3. La fenêtre d'installation de NetBeans s'affiche. Cliquez sur Continuer.

Figure A-29

Copyright © 2016 Eyrolles.

4. Le panneau suivant vous propose de sélectionner le disque sur lequel seront installées les applications JDK et NetBeans. Cliquez sur le bouton Continuer.

Figure A-30

5. Une fenêtre indiquant le mode d'installation et l'espace requis par NetBeans apparaît. Cliquez sur le bouton Installer pour lancer l'installation.

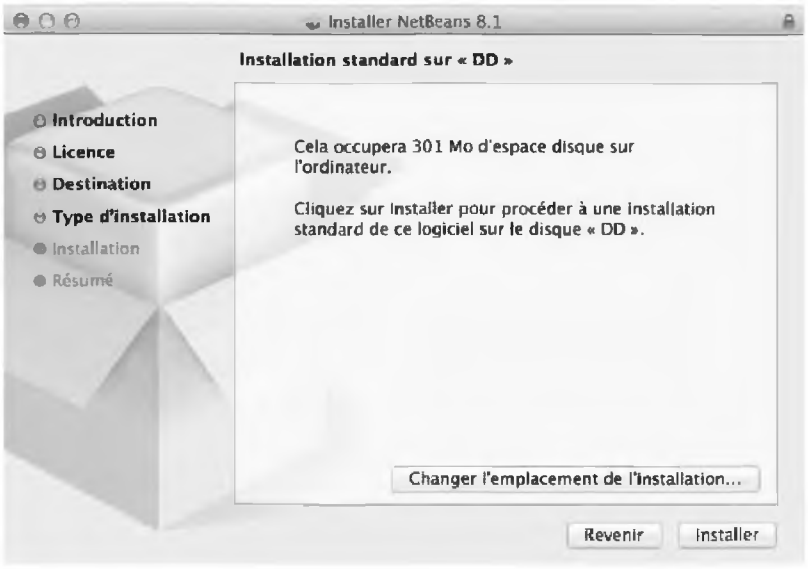


Figure A-31

Copyright © 2016 Eyrolles.

6. L'installation démarre et son état d'avancement est indiqué par une barre de progression.

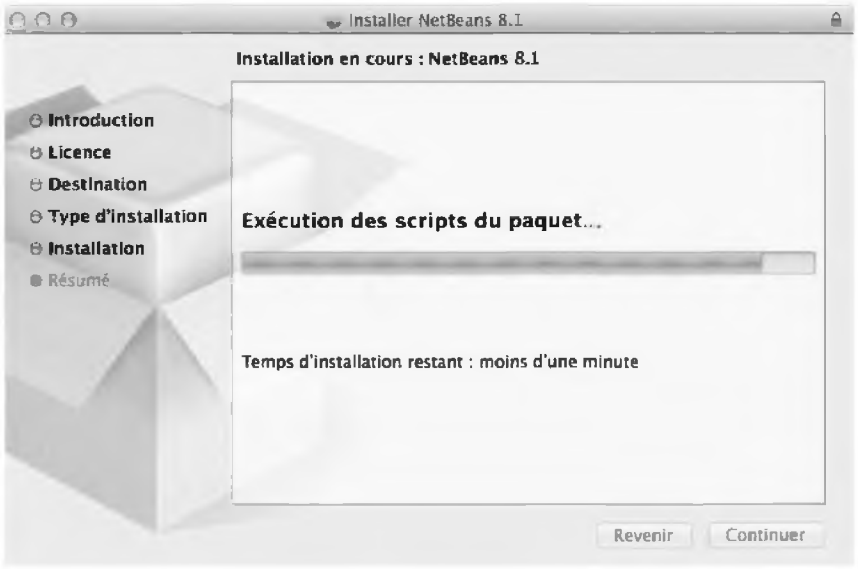


Figure A-32

7. Une fois l'installation terminée, cliquez sur le bouton Fermer. Il n'est pas nécessaire de redémarrer votre ordinateur.

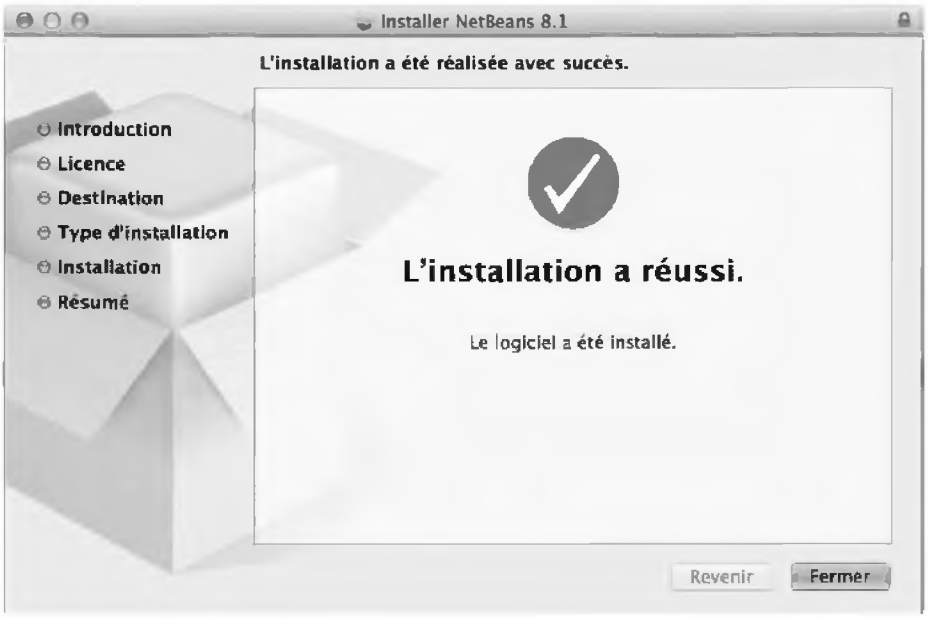


Figure A-33

Copyright © 2016 Eyrolles.

8. Éjectez le paquet `NetBeans 8.1.pkg` et supprimez le fichier `netbeans-8.1-javase-macosx.dmg`.

Lancement de l'environnement NetBeans

Pour lancer NetBeans, déplacez-vous dans le répertoire `Applications/NetBeans`. Double-cliquez sur l'icône `NetBeans 8.1`.

Remarque Pour simplifier le lancement de NetBeans, vous pouvez déplacer l'icône `NetBeans 8.1` vers le Dock.

Après avoir double-cliqué sur l'icône, l'environnement NetBeans démarre et la fenêtre de présentation de l'application apparaît.

Figure A-34

Une fois lancé, NetBeans affiche à l'écran un ensemble de fenêtres vides ainsi qu'une page d'accueil.

La description et l'utilisation de l'environnement de développement d'applications Java seront décrites à la section « Utilisation des outils de développement – Développer avec NetBeans ».

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

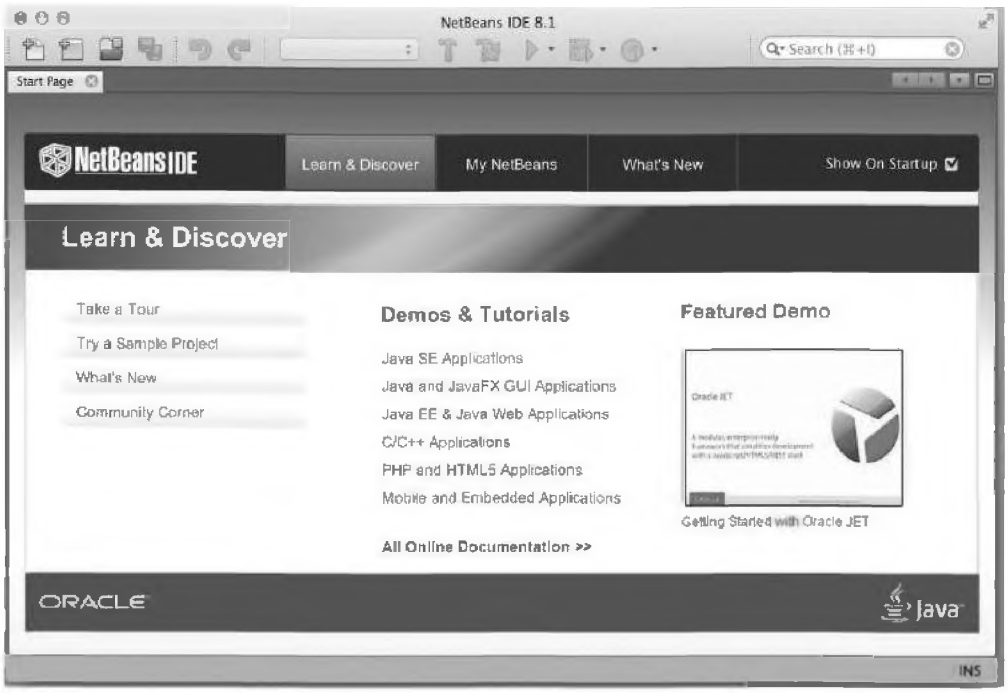


Figure A-35

Charger le plug-in JUnit et/ou Hamcrest

Le plug-in JUnit n'est pas intégré par défaut dans le module d'installation de NetBeans 8.1. Vous devez l'installer manuellement en suivant la procédure suivante.

- Sous NetBeans, sélectionnez l'item Open Project... du menu File. Le panneau Open Project s'ouvre, parcourez l'arborescence des exemples de l'extension Web et choisissez le projet Sources/Exemple/Chapitre12/ NetBeansProjects/Cercle.

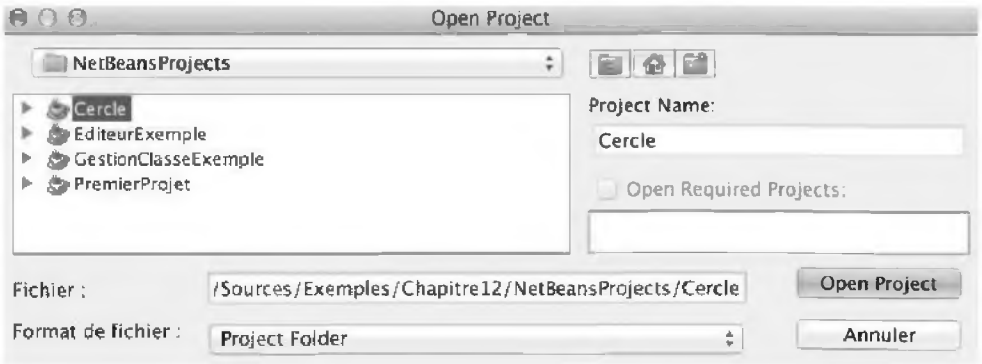


Figure A-36

Copyright © 2016 Eyrolles.

- À l'ouverture du projet Cercle, l'application Netbeans indique qu'il y a un problème qu'il est possible de résoudre en cliquant sur le bouton Resolve Problems…

Figure A-37

- Le panneau Resolve Project Problems suivant s'affiche, cliquez sur le bouton Resolve…

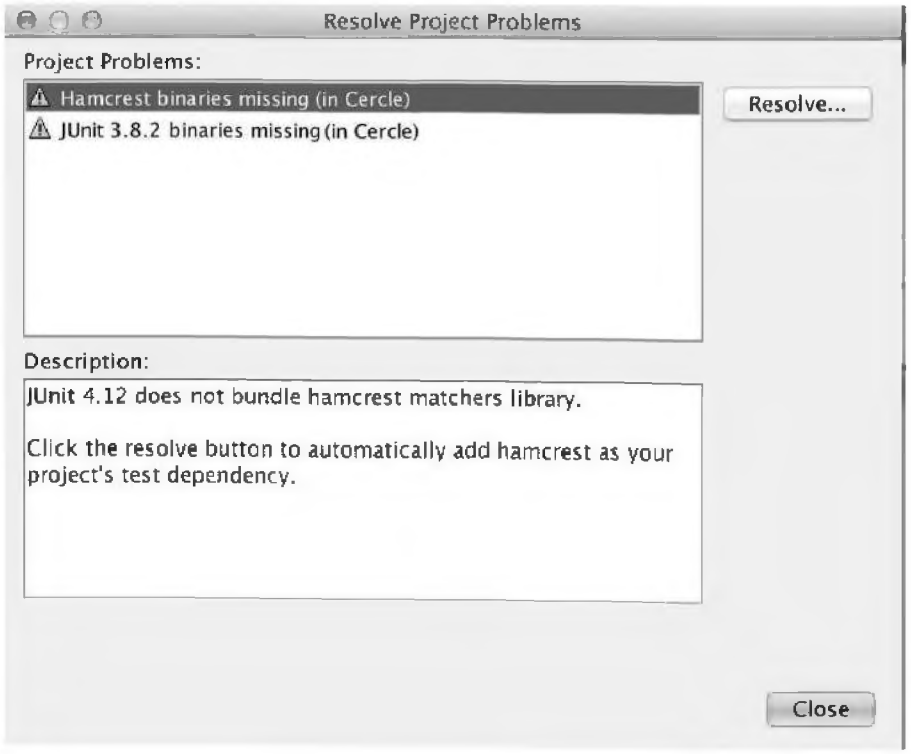


Figure A-38

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

- L'interface Netbeans trouve les solutions et installe les plug-ins requis. Le panneau Resolve Project Problems affiche les résultats des corrections apportées.

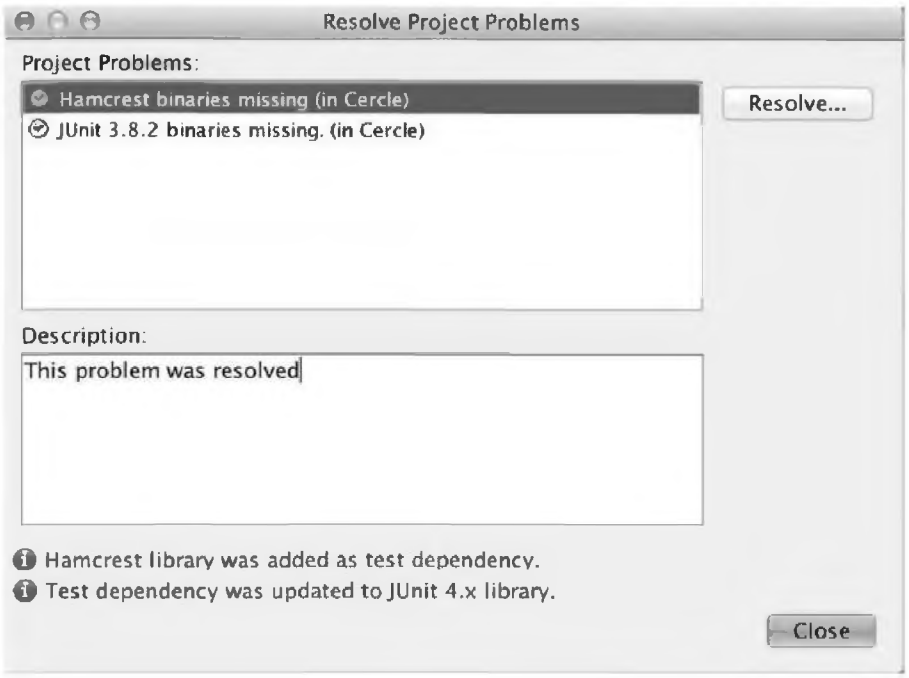


Figure A-39

Installation de NetBeans sous Linux

Une fois le fichier d'installation `netbeans-8.1-javase-linux.sh` téléchargé depuis l'extension Web, déplacez-le sur le Bureau de votre ordinateur. L'installation de NetBeans s'effectue sous le compte utilisateur `root`.

Remarque Sous Ubuntu, vous trouverez directement le paquet `netbeans` dans le gestionnaire de paquets.

L'installation

1. Copiez le fichier `netbeans-8.1-javase-linux.sh` dans le répertoire `/tmp/installation` et déplacez-vous dans le répertoire `/usr/local` à l'aide de la commande :

```
cd /usr/local.
```

2. Lancez l'installation en tapant la commande :

```
/tmp/installation/netbeans-8.1-javase-linux.sh
```

Copyright © 2016 Eyrolles.

L'exécutable lance l'installation et affiche :

```
Configuring the installer...
Searching for JVM on the system...
Extracting installation data...
Running the installer wizard...
```

3. Suivez la procédure d'installation en cliquant sur le bouton Suivant.

Figure A-40

4. Acceptez les termes de la licence en cochant l'option J'accepte les termes..., puis cliquez sur le bouton Suivant.

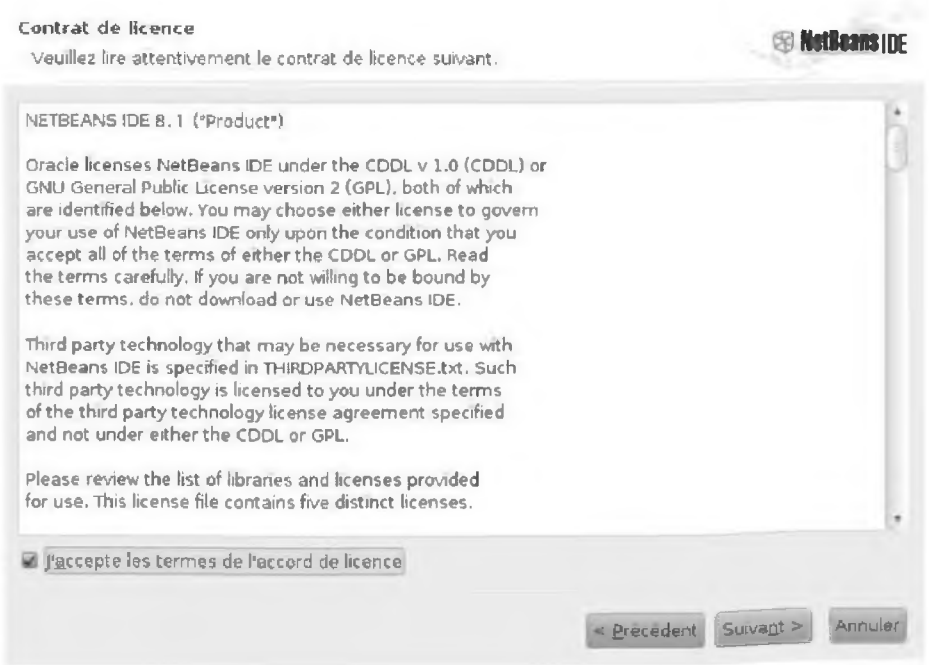


Figure A-41

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

5. Acceptez l'installation du framework JUnit en cliquant sur le bouton Suivant.

Figure A-42

6. Dans la fenêtre suivante, modifiez si vous le souhaitez le répertoire d'installation de NetBeans, puis cliquez sur Installer.
7. Une nouvelle fenêtre vous indique ensuite que l'installation est terminée. Cliquez sur le bouton Terminer. Il n'est pas nécessaire de redémarrer votre ordinateur.

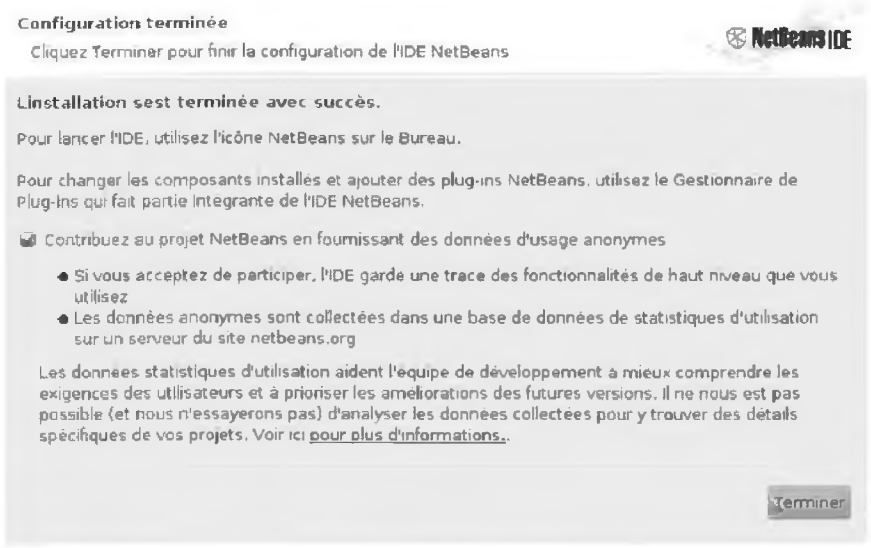


Figure A-43

Copyright © 2016 Eyrolles.

8. Supprimez le fichier d'installation `netbeans-8.1-ml-javase-linux.sh`.
9. Pour simplifier l'appel à NetBeans, créez un lien symbolique vers `netbeans-8.0.2` grâce à la commande suivante :

```
ln -s netbeans-8.1 netbeans
```

10. Modifiez le fichier `/home/votreCompte/.bashrc` de votre compte utilisateur en y ajoutant la ligne suivante :

```
PATH=$PATH :/usr/local/netbeans/bin; export PATH
```

11. Exécutez votre profil `.bashrc` en tapant la commande :

```
. .bashrc
```

Remarque Sous Ubuntu, cliquez sur l'icône `Ide NetBeans 8.0.2` qui apparaît sur le Bureau en fin d'installation.

Lancement de l'environnement NetBeans

Pour lancer NetBeans, il suffit de taper la commande suivante dans une fenêtre de commandes :

```
netbeans &
```

L'environnement NetBeans s'exécute et affiche la fenêtre de présentation de l'application.

Figure A-44

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

Une fois lancé, NetBeans affiche à l'écran un ensemble de fenêtres vides, ainsi qu'une page d'accueil.

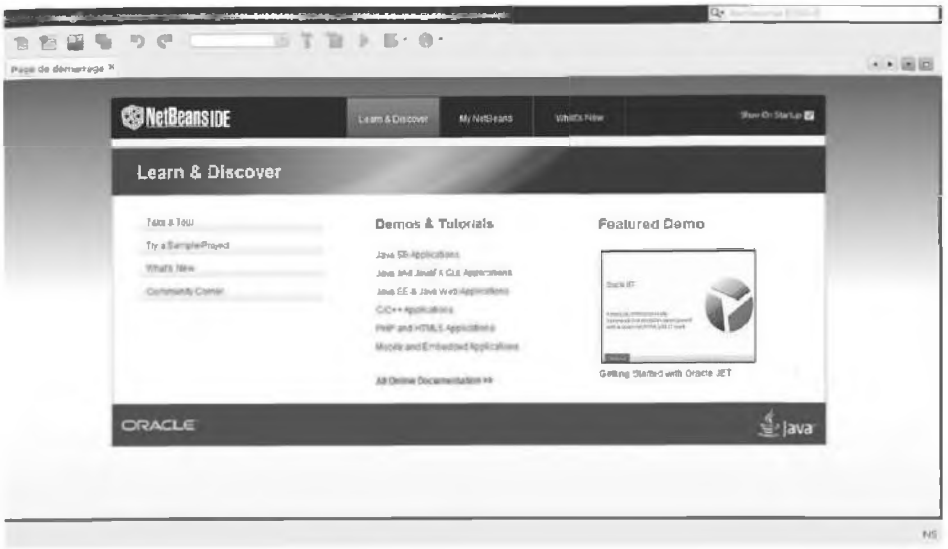


Figure A-45

La description et l'utilisation de l'environnement de développement d'applications Java seront décrites à la section « Utilisation des outils de développement – Développer avec NetBeans » ci-après.

Utilisation des outils de développement

Installer la documentation en ligne

La documentation en ligne (*http://download.oracle.com/javase/8/docs/api*) est très utile lorsqu'on développe des applications Java. Elle décrit l'ensemble des outils de développement proposés par le langage.

Développer en mode commande

Le développement d'applications passe par trois étapes qui se répètent tant que le programme ne répond pas aux attentes du programmeur. Ces trois étapes sont :

1. L'édition, qui consiste à écrire un programme à l'aide d'un éditeur de texte. Le fichier porte le nom de la classe qu'il définit, suivi de l'extension `.java`.
2. La compilation, qui permet la traduction du programme en un format compréhensible par l'ordinateur. Si la compilation est exécutée sans avoir détecté d'erreurs, le fichier exécutable porte le même nom que celui du programme, suivi de l'extension `.class`.
3. L'exécution, qui permet de vérifier le bon fonctionnement du programme.

Copyright © 2016 Eyrolles.

S'il subsiste des erreurs ou des incohérences en phase de compilation ou lors de l'exécution, il est nécessaire de corriger ou de modifier le programme à l'aide de l'éditeur de texte. Une fois les corrections ou modifications réalisées, le programmeur lance à nouveau la compilation, puis l'exécution.

Pour éviter d'avoir à répéter les opérations d'ouverture du fichier contenant le programme avec un éditeur de texte, de sortie de l'éditeur avec sauvegarde, de lancement de commandes de compilation ou d'exécution du programme..., il convient d'ouvrir deux fenêtres de travail : une pour éditer le programme, l'autre pour lancer les commandes de compilation ou d'exécution.

Les fenêtres de commandes

Les fenêtres de commandes sont utilisées pour lancer les commandes de compilation et d'exécution des programmes Java.

Pour compiler un programme Java, la commande est la suivante :

```
javac NomDeLaClasse.java
```

où `NomDeLaClasse` correspond au nom du fichier (veillez à bien respecter l'orthographe, ainsi que les majuscules et les minuscules). L'utilisation de l'extension `.java` est obligatoire.

Pour exécuter un programme Java, la commande est la suivante :

```
java NomDeLaClasse
```

Aucune extension ne doit être placée à la fin de la ligne de commande.

cmd.exe sous Windows 2000, XP, Vista et 7

Sous Windows 2000 et versions ultérieures, la fenêtre de commandes s'ouvre à l'appel du programme `cmd.exe`. Voici comment appeler ce programme :

- Sous Windows 2000 et XP :
  - Dans le menu Démarrer, sélectionnez Exécuter.
  - Tapez la commande `cmd` dans la boîte de dialogue.
- Sous Windows Vista et 7 :
  - Dans le menu Démarrer, cliquez dans la zone de texte Rechercher située en bas à gauche du menu.
  - Tapez la commande `cmd` dans la boîte de dialogue.

Une fenêtre sur fond noir apparaît. Il s'agit de la fenêtre Invite de commandes.

Les commandes sont lancées par défaut à partir du répertoire `C:\Documents and Settings\NomUtilisateur` (Windows 2000 à XP) et `C:\Users\NomUtilisateur` (Windows Vista et 7). Pour que la fenêtre s'ouvre directement à partir du répertoire dans lequel sont placées vos applications Java, il convient de modifier ses paramètres.

Modifier les paramètres de la fenêtre Invite de commandes

Pour modifier les paramètres de la fenêtre Invite de commandes, rendez-vous dans le répertoire `C:\Windows\System32`, puis effectuez un clic droit sur l'icône `cmd.exe` et choisissez Créer un raccourci. Déplacez le raccourci sur le Bureau et cliquez droit dessus. Dans le menu contextuel qui apparaît, sélectionnez Propriétés pour afficher la fenêtre suivante.

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

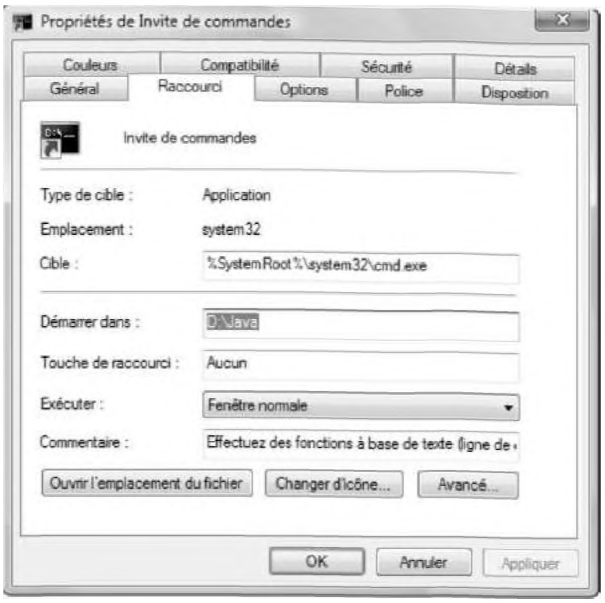


Figure A-46

Cliquez sur l'onglet Raccourci et dans le champ Démarrer dans, saisissez le chemin d'accès au répertoire de travail soit, par exemple, `D:\java`.

Les autres onglets vous permettront, par exemple, de modifier la taille et la couleur de la fenêtre, ainsi que celles de la police de caractères.

Validez les modifications effectuées en cliquant sur le bouton OK.

Terminal sous Mac OS X et Linux

Sous Mac OS X et Linux, la fenêtre de commandes s'appelle un terminal.

- Sous Mac OS X, double-cliquez sur l'icône `Terminal` du répertoire `Applications/Utilitaires`.
- Sous Linux, ouvrez une fenêtre Terminal en sélectionnant `Terminal` dans `Applications/Accessoires`. Déplacez-vous ensuite dans le répertoire où sont enregistrés vos programmes Java avec la commande suivante :

```
cd nomDuDossier
```

Créer un environnement de travail

Pour créer un environnement de travail, vous devez ouvrir une « fenêtre d'édition » pour écrire vos programmes et « une fenêtre de commandes » pour les compiler et les exécuter.

Écrire un programme

Lancer un éditeur de texte de type Bloc-notes (Windows), emacs, vi, vim (Linux) ou encore TextEdit (Mac). Au moment d'enregistrer votre fichier, veillez à ce que le format texte soit bien sélectionné.

Copyright © 2016 Eyrolles.

- Dans le Bloc-notes, allez dans le menu Fichier>Enregistrer. La première fois que vous enregistrez le programme, l'éditeur demande le nom du fichier à sauvegarder. Veillez à sélectionner Tous les fichiers dans liste déroulante Type afin de pouvoir enregistrer le fichier avec l'extension `.java`.

Figure A-47

- Dans TextEdit, modifiez les préférences du logiciel en sélectionnant via le menu Text Edit>Préférences. Dans la fenêtre Préférences, cochez l'option Format Texte. Enregistrez ensuite le fichier au format UTF-8.

Figure A-48

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

Vous n'avez pas besoin de sortir de l'éditeur pour compiler et exécuter votre programme ; il suffit de ne pas oublier de l'enregistrer après chaque modification.

Compiler et exécuter un programme

Ouvrez une fenêtre Invite de commandes (en cliquant sur le raccourci créé précédemment) ou un Terminal, selon votre système d'exploitation (respectivement Windows ou Mac, Linux).

Cette fenêtre sert de fenêtre de compilation et d'exécution des programmes écrits à partir de la « fenêtre d'édition ».

- Compilez le programme écrit à l'étape précédente en tapant la commande `javac` suivie du nom du fichier et de l'extension `.java`, par exemple `javac Cercle.java`.
- Si des messages d'erreur de compilation apparaissent, notez la première erreur avant de retourner à la « fenêtre d'édition ». Corrigez l'erreur, puis enregistrez le fichier pour prendre en compte la correction.
- Retournez dans la fenêtre de commandes et recompilez jusqu'à ce qu'il n'y ait plus d'erreurs.
- Lorsque votre programme est correct (il n'y a plus d'erreurs de compilation), exécutez-le en tapant la commande `java` suivie du nom du programme sans extension, par exemple `java Cercle`.

Créer un répertoire commun

Lorsque l'on commence à écrire des applications utilisant plusieurs fichiers, il est intéressant d'utiliser la variable d'environnement `CLASSPATH`.

En effet, certains fichiers (classes) sont utilisés par différents programmes. En plaçant ces fichiers dans un dossier appelé, par exemple, `commun` et en initialisant la variable `CLASSPATH` à `Un/répertoire/commun`, nous indiquons au compilateur comment il doit rechercher les classes qui ne sont pas définies dans le répertoire où se trouve le fichier qu'il compile.

Sous Windows 2000, NT, XP, Vista et 7

La définition de la variable `CLASSPATH` s'effectue par l'intermédiaire de l'outil Système défini dans le Panneau de configuration.

En suivant les instructions de la section « Installation de J2SE SDK 7 sous Windows – Les variables d'environnement », modifiez la variable système `CLASSPATH` et ajoutez, par exemple, la valeur `D:\Java\commun`.

Copyright © 2016 Eyrolles.

Sous Linux

La définition de la variable `CLASSPATH` s'effectue en modifiant le fichier `/home/votre Compte/.bashrc` de votre compte utilisateur, en y ajoutant les lignes suivantes :

```
CLASSPATH =\leRepertoire\ouSeTrouvent\lesDifferentes\classes\
aCompiler\.
export CLASSPATH
```

Exécutez votre profil `.bashrc` en tapant la commande :

```
. .bashrc
```

Exemple

Le fichier `Compte.java` est un bon exemple de programme utilisé par des applications différentes (voir à partir du chapitre 10, « Collectionner un nombre indéterminé d'objets », section « Le projet : gestion d'un compte bancaire »). Pour éviter d'avoir à copier le fichier dans le répertoire où se trouve l'application à développer, il est plus judicieux de le placer une seule fois dans le dossier `D:\Java\commun` ou `/home/votreCompte/Java/commun` selon votre environnement.

De cette façon, lors de la compilation d'un programme utilisant un objet de type `Compte`, le compilateur ne trouvant pas le fichier `Compte.java` dans le répertoire de l'application ira le chercher automatiquement dans le répertoire `commun`.

Développer avec NetBeans

Pour le lecteur débutant, nous conseillons vivement de commencer à développer vos programmes Java en utilisant le mode commande avant d'utiliser NetBeans. En effet, avec NetBeans, les programmes sont compilés au fur et à mesure que vous les écrivez. Cela facilite bien évidemment le développement des applications, mais cela ne permet pas aux débutants de bien maîtriser toutes les étapes de développement d'une application, notamment la partie découverte des messages d'erreurs et leur correction.

Créer un projet Java

Sous NetBeans, toute application Java doit être écrite au sein d'un projet. Pour créer un projet, sélectionnez le menu Fichier>Nouveau Projet de NetBeans.

Dans la boîte de dialogue qui apparaît, choisissez la catégorie Java et comme type de projet Application Java. Cliquez ensuite sur le bouton Suivant.

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

Figure A-49

Une nouvelle boîte de dialogue apparaît, nommée Nouveau Application Java (voir figure A-49).

1. Entrez le nom du projet en première ligne (ici, `Cercle`).
2. Spécifiez le répertoire d'enregistrement du projet en cliquant sur le bouton Parcourir, ou conservez les valeurs par défaut proposées par NetBeans.
3. Créez la classe principale, nommée ici `Introduction.Cercle`. Le premier terme `Introduction` indique le nom du package au sein duquel est enregistrée la classe `Cercle.java`.

Remarque

La notion de package est abordée à la section « Où se trouvent les programmes créés sous NetBeans ? » ci-après.

Copyright © 2016 Eyrolles.

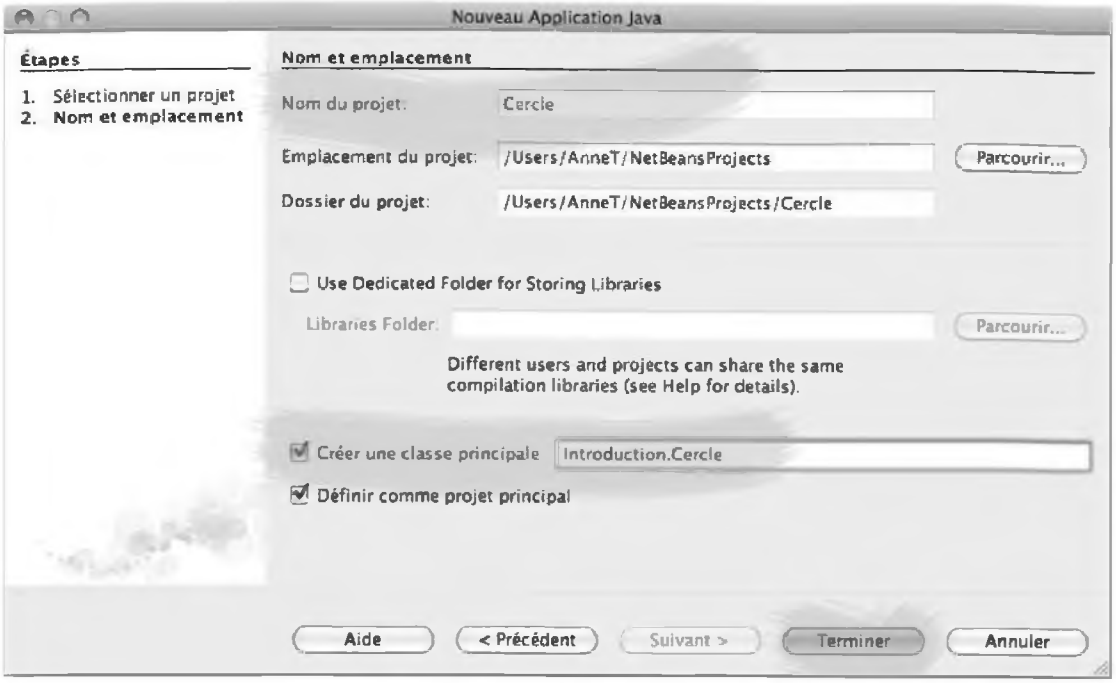


Figure A-50

4. Cliquez enfin sur le bouton Terminer pour créer et enregistrer le projet. NetBeans affiche alors un ensemble de panneaux (voir figure A-51), dont celui correspondant à la classe `Cercle`.

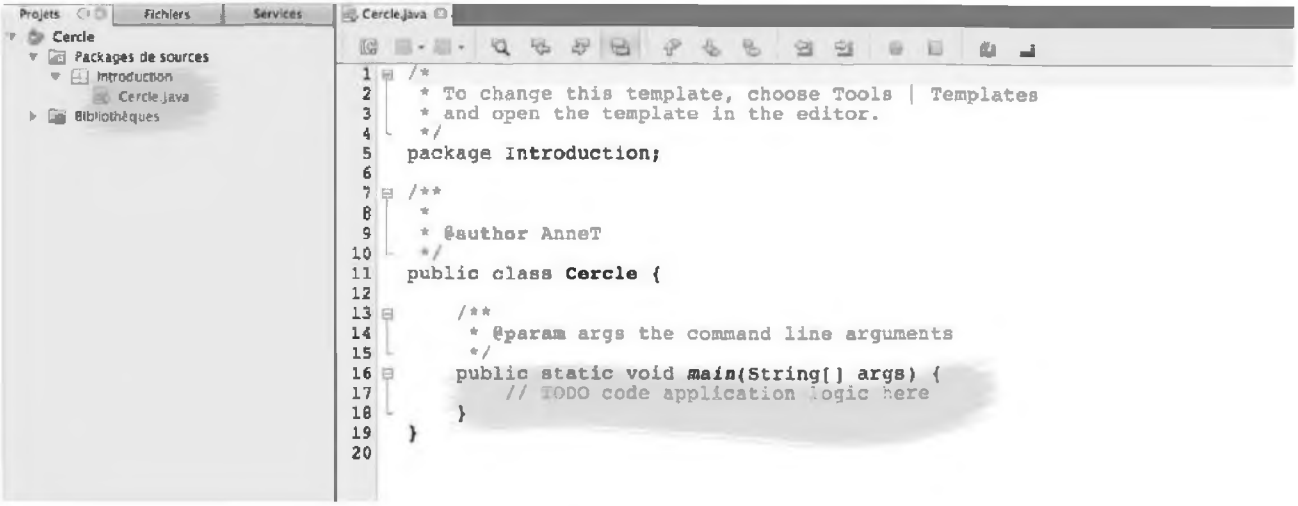


Figure A-51

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

Le panneau Projets, situé dans la partie gauche de l'environnement, indique les noms des fichiers inclus dans le projet en cours. Le panneau central constitue le panneau principal dans lequel vous allez écrire votre programme.

Écrire un premier programme Java

Comme le montre la figure A-51, la fenêtre d'édition s'affiche au centre de l'espace de travail avec quelques lignes de code. NetBeans a écrit pour vous le squelette général de la classe `Cercle` avec sa fonction `main()`, à vous de la compléter ensuite.

Si vous le souhaitez, vous pouvez supprimer les lignes de commentaires proposées par NetBeans.

Nous vous proposons de recopier la classe `Cercle` décrite au chapitre introductif, « Naissance d'un programme », section « Un premier programme Java ».

Lorsque vous saisissez les instructions observez que :

- si vous faites une faute, NetBeans vous la signale immédiatement en la soulignant d'un trait rouge et en affichant une marque sur le côté gauche de la fenêtre d'édition. Si vous passez le curseur de la souris sur cette marque, une infobulle apparaît avec la raison de l'erreur (voir figure A-52) ;
- lorsque vous saisissez le signe . (point), par exemple après `System`, et que vous marquez une pause, NetBeans propose une liste dans laquelle vous pouvez choisir un nom de méthode ou de variable ;

```
2   package cercle;
3
4   /*
5   #       Le livre de Java 1er langage
6   #       A. Tasso
7   #       Introduction : Naissance d'un programme
8   #       Section  : Un premier programme Java
9   #       Fichier  : Cercle.java
10  #       Class    : Cercle
11  */
12  import java.util.*;
13  public class Cercle     {
14   public static void main(String [] argument)  {
15          double unRayon, lePerimetre;
16          Scanner lectureClavier = new Scanner(System.in);
17          nt("Valeur du rayon : ");
18          ureClavier.nextDouble();
19          2 * Math.PI * unRayon;
            lectureClavier.
            System.out.print("Le cercle de rayon " + unRayon);
22          System.out.println(" a pour perimetre : "+ lePerimetre);
23   }
24  }
```

cannot find symbol
symbol : variable System
location: class java.util.Scanner

Figure A-52

Copyright © 2016 Eyrolles.

- lorsque vous cliquez sur une variable ou sur un nom de méthode, NetBeans surligne en jaune toutes ses occurrences dans le fichier ;
- la sauvegarde du programme (Ctrl+S) entraîne la compilation de l'application.

Exécuter une application Java

Pour exécuter une application, cliquez sur l'icône représentant une flèche verte située au centre de la boîte à outils de NetBeans ou tapez sur la touche F6 de votre clavier.

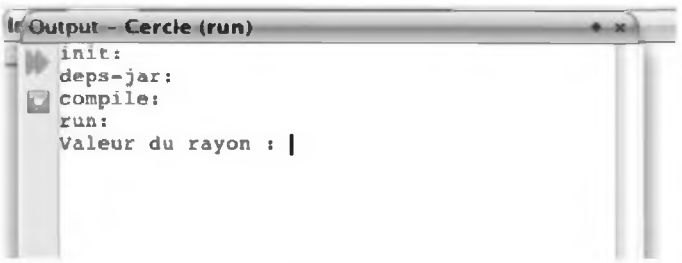


Figure A-53

L'exécution de l'application `Cercle` s'effectue dans la fenêtre Sortie qui apparaît au moment de l'exécution (voir figure A-53). Il est possible de :

- stopper l'exécution de l'application avant qu'elle ne se termine d'elle-même, en cliquant sur l'icône représentant un rectangle rouge située sur le côté gauche de la fenêtre Sortie ;
- relancer l'exécution de l'application en cliquant sur les deux flèches vertes situées sur le côté gauche de la fenêtre Sortie.

Où se trouvent les programmes créés sous NetBeans ?

Lorsque vous créez un projet sous NetBeans, l'application crée d'elle-même un répertoire portant le nom du projet, sous un répertoire de travail nommé `NetBeansProjects`. Ce dernier est créé par défaut dans votre répertoire personnel.

Il est possible de modifier le dossier d'enregistrement du projet en cliquant sur le bouton Parcourir de la fenêtre Nouveau Application Java (voir figure A-50).

NetBeans crée ensuite, à partir du répertoire racine du projet (par exemple, `.../NetBeans Projects/Cercle`), une arborescence de fichiers contenant notamment les répertoires `build` et `src`.

- Le répertoire `build` contient tous les fichiers compilés du projet, c'est-à-dire ceux d'extension `.class`.
- Le répertoire `src` contient tous les fichiers sources du projet, c'est-à-dire ceux d'extension `.java`.

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

Ces fichiers sont enregistrés sous chacun des répertoires `build` et `src`, dans un sous-répertoire dont le nom correspond au nom du package au sein duquel ils ont été créés.

Ainsi, par exemple, le fichier `Cercle.java` défini au sein du projet `Cercle` et du package `introduction` se trouvera dans le répertoire `D:\NetBeansProjects\Cercle\src\Introduction` ou `/home/votreCompte/NetBeansProjects/Cercle/src/Introduction` selon votre environnement.

Rien ne vous interdit d'utiliser ces fichiers pour les compiler ou les exécuter avec un autre outil de développement.

Importer un fichier Java

À l'inverse, vous pouvez copier des fichiers de classe développée par ailleurs, dans un répertoire correspondant à un projet NetBeans. Il convient alors de :

- placer les fichiers d'extension `.java` dans le répertoire `nomDuProjet/src/nomDuPackage` ;
- insérer au tout début des fichiers importés l'instruction définissant le nom du package où ils sont enregistrés, par exemple :

```
package Introduction;
```

Regénérer un projet Java (refactoring)

Il est possible de modifier le nom d'une classe appartenant à un projet en cours de développement.

1. Effectuez un clic droit sur le nom de la classe à modifier dans la fenêtre Projets.
2. Dans le menu contextuel qui apparaît, sélectionnez Refactorer, puis Renommer.

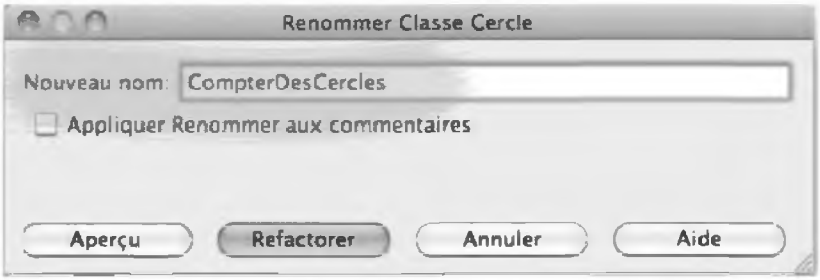


Figure A-54

3. Dans la fenêtre Renommer NomDeLaClasse qui s'affiche, saisissez le nouveau nom de la classe et cliquer sur le bouton Refactorer.

Toutes les classes du projet qui utilisaient l'ancien nom de la classe vont être automatiquement modifiées.

Copyright © 2016 Eyrolles.

Grâce au *refactoring*, il est également possible de copier une classe d'un projet vers un autre projet.

1. Effectuez un clic droit sur le nom de la classe dans la fenêtre Projets.
2. Dans le menu contextuel qui s'ouvre, sélectionnez Copier.
3. Dans la fenêtre Projets, effectuez un clic droit sur le nom du package associé au projet au sein duquel vous souhaitez copier la classe.
4. Dans le menu contextuel, sélectionnez Coller, puis Refactorer Copier.
5. Modifiez éventuellement le nom de la classe à copier et cliquez sur le bouton Refactorer.

La classe fait ensuite intégralement partie du projet et du package correspondant.

Développer des applications Android avec Android Studio

Grâce à l'IDE Android Studio proposé par Google, il est possible de développer des applications Android de façon conviviale.

Installer l'IDE Android Studio

Une fois le fichier d'installation `android-studio-ide` correspondant à votre système d'exploitation téléchargé depuis l'extension Web, déplacez-le sur le Bureau de l'ordinateur.

L'installation

1. Double-cliquez sur le fichier d'installation, une fenêtre d'installation Android Studio Setup apparaît, cliquez sur Next.

Figure A-55

Copyright © 2016 Eyrolles.

2. La fenêtre d'installation vous propose de choisir les composants que vous souhaitez installer. Choisissez les composants à installer par défaut en cliquant sur Next.

Figure A-56

3. Acceptez les termes de la licence en cliquant sur le bouton I Agree.

Figure A-57

Copyright © 2016 Eyrolles.

4. Vous pouvez changer le répertoire d'installation de l'environnement en cliquant sur le bouton Browse...
5. Cliquez sur le bouton Next pour lancer l'installation.

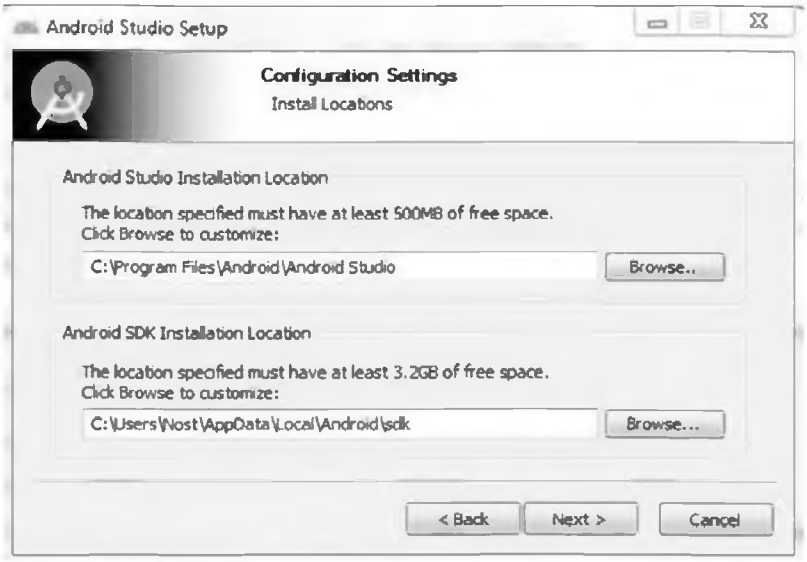


Figure A-58

6. Vous pouvez accélérer les performances de l'émulateur proposé par l'IDE en augmentant la mémoire allouée par l'application. Un minimum de 2 Go est recommandé. Cliquez sur Next.

Figure A-59

Copyright © 2016 Eyrolles.

7. Après avoir choisi votre répertoire de lancement de l'application, dans le menu Démarrer, cliquez sur Install.

Figure A-60

8. Le programme d'installation démarre, cliquez sur le bouton Next lorsque l'installation est terminée.

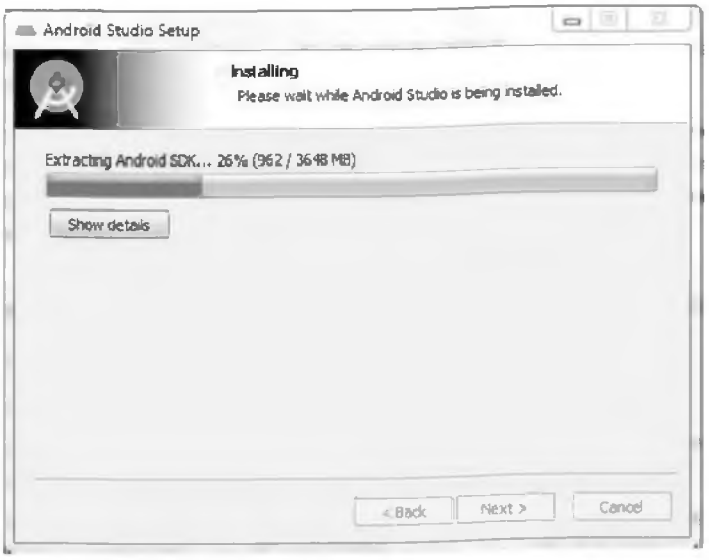


Figure A-61

Copyright © 2016 Eyrolles.

9. L'installation est terminée, vous pouvez cliquer sur Finish.

Figure A-62

10. L'application Android Studio lancée en fin d'installation propose d'importer les paramètres d'une éventuelle version précédente. Sélectionnez la puce correspondant à votre situation et cliquez sur OK.

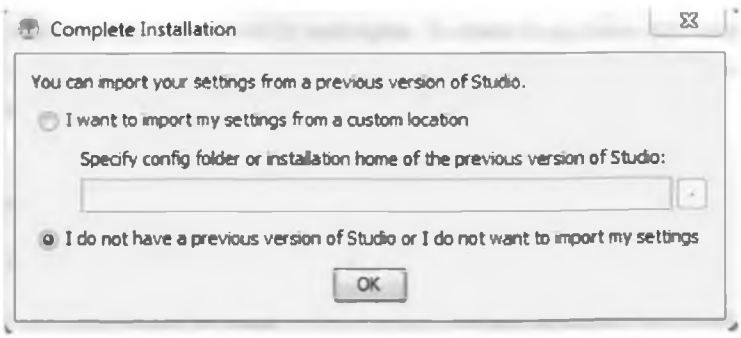


Figure A-63

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

11. L'application met à jour un certain nombre de composants. Cette mise à jour prend quelques minutes. Elle est terminée lorsque vous pouvez cliquer sur le bouton Finish.

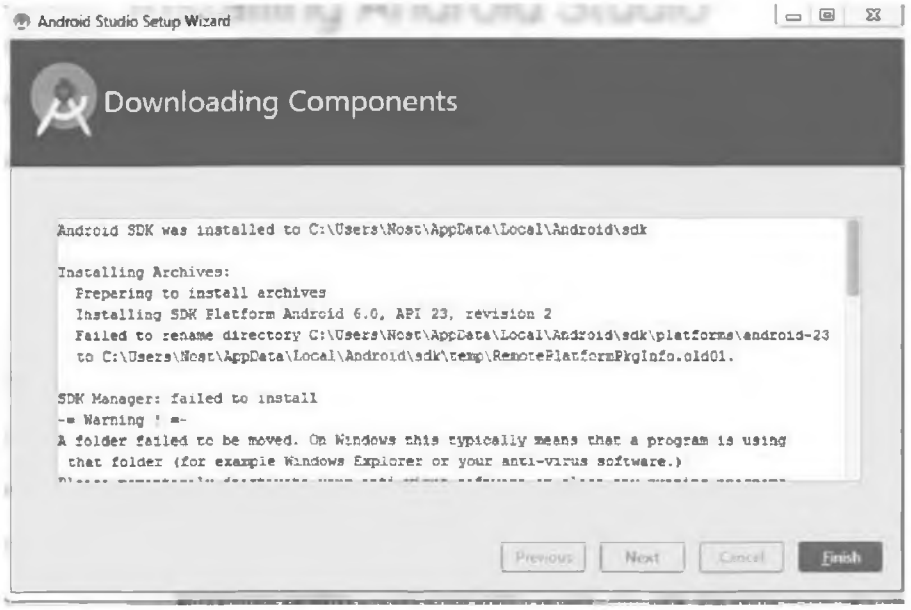


Figure A-64

Télécharger les SDK Android via Android Studio

Une fois installée et lancée, l'application Android Studio présente un écran d'accueil comme ci-après (voir figure A-65). Pour accéder à l'IDE de programmation, sélectionnez l'item « Start a new Android Studio project ».

Remarque

Tout comme avec l'IDE NetBeans, le développement d'applications mobiles sous Android Studio s'effectue en mode projet. La création d'un projet sous Android Studio est décrite chapitre 13 « Développer une application mobile », section « Bonjour le monde : votre première application mobile ».

Android Studio est constitué d'une fenêtre d'édition du code avec coloration syntaxique et surtout d'un éditeur graphique représentant le dispositif (smatphone, tablette, TC…) pour lequel vous souhaitez développer une application (voir figure A-66). Vous pouvez passer en mode Text et/ou Design en cliquant sur l'onglet (situé en bas au centre) correspondant.

Pour développer une application sur un dispositif en particulier, vous devez télécharger le SDK correspondant, via Android Studio. Pour cela, rendez-vous dans le menu Tools et sélectionnez Android SDK Manager.

Copyright © 2016 Eyrolles.

Figure A-65

Remarque Un SDK (*Software Development Kit* ou Kit de développement logiciel) est une boîte à outils qui facilite le développement d'applications en proposant des bibliothèques de fonctions spécifiques au matériel (tablette ou smartphone) sur lequel elles seront exécutées.

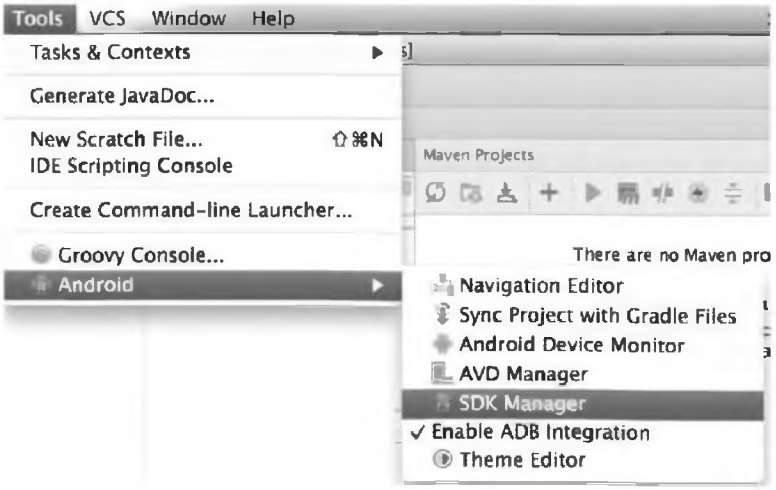


Figure A-66

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

1. Le panneau Default Settings apparaît, cochez les cases correspondantes au SDK que vous souhaitez installer (ici Android 6.0 et Android 2.2).

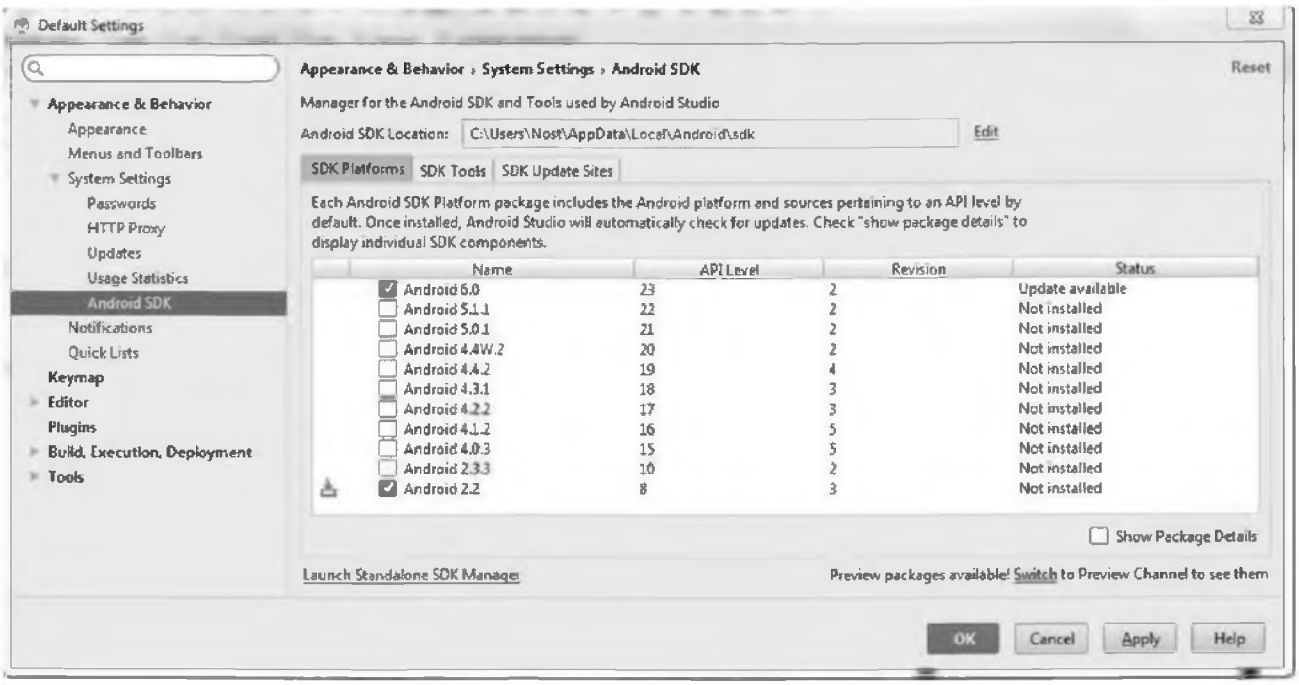


Figure A-67

2. Patientez un certain temps (entre dix et vingt minutes). Lorsque la fenêtre confirmant l'installation des différents SDK apparaît, il est temps de créer votre propre émulateur.

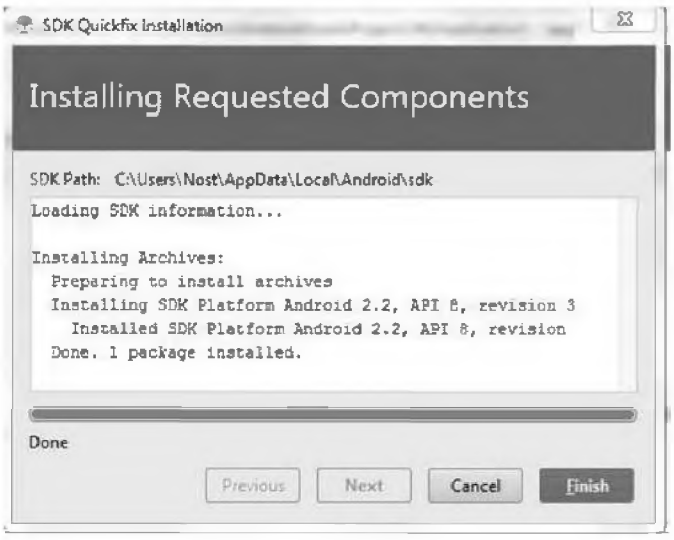


Figure A-68

Copyright © 2016 Eyrolles.

Créer un émulateur Android

Une fois les SDK installés, il convient de créer un émulateur Android afin de vérifier le bon fonctionnement des applications que vous développerez sous Android Studio. Grâce à cet émulateur, vous testerez vos applications sans avoir besoin de connecter un dispositif Android (smartphone, tablette…) à l'ordinateur.

Pour mettre en place l'émulateur Android, sélectionnez AVD Manager dans le menu Tools - Android.

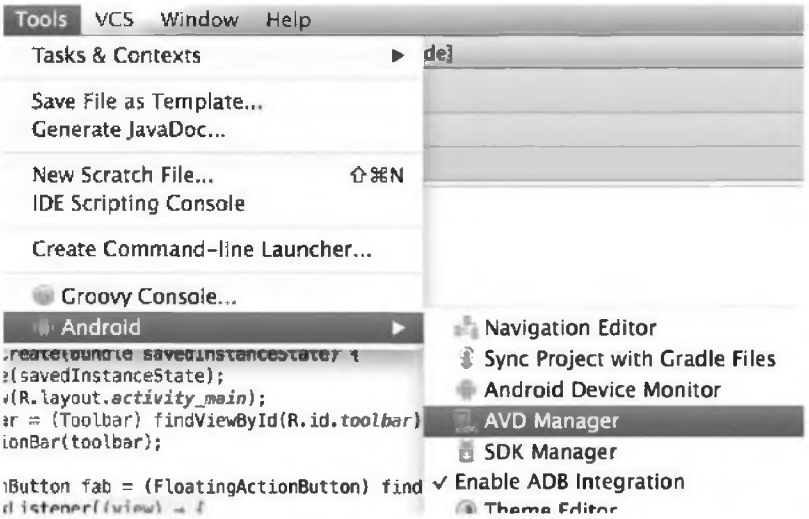


Figure A-69

1. Le panneau AVD Manager apparaît. Pour créer votre propre émulateur, cliquez sur Create Virtual Device…

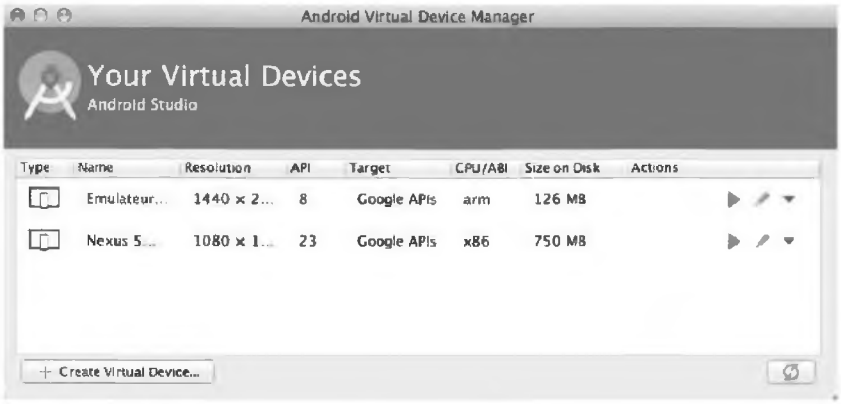


Figure A-70

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

2. Le panneau Virtual Device Configuration apparaît. Sélectionnez le smartphone que vous souhaitez émuler (ici Nexus S), puis cliquez sur Next.

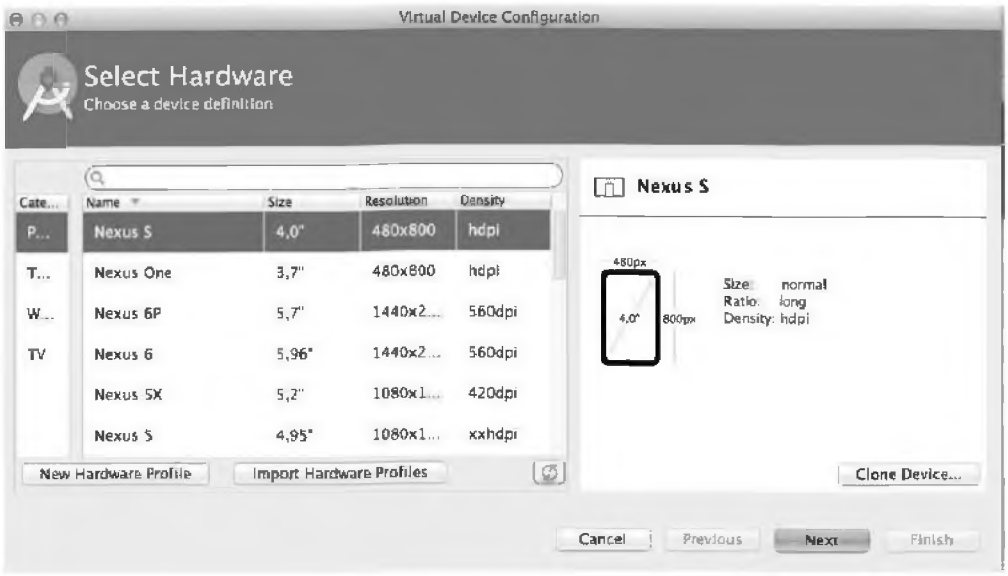


Figure A-71

3. Choisissez l'image system, sachant que celle correspondant à la release Marshmallow est la mieux adaptée.

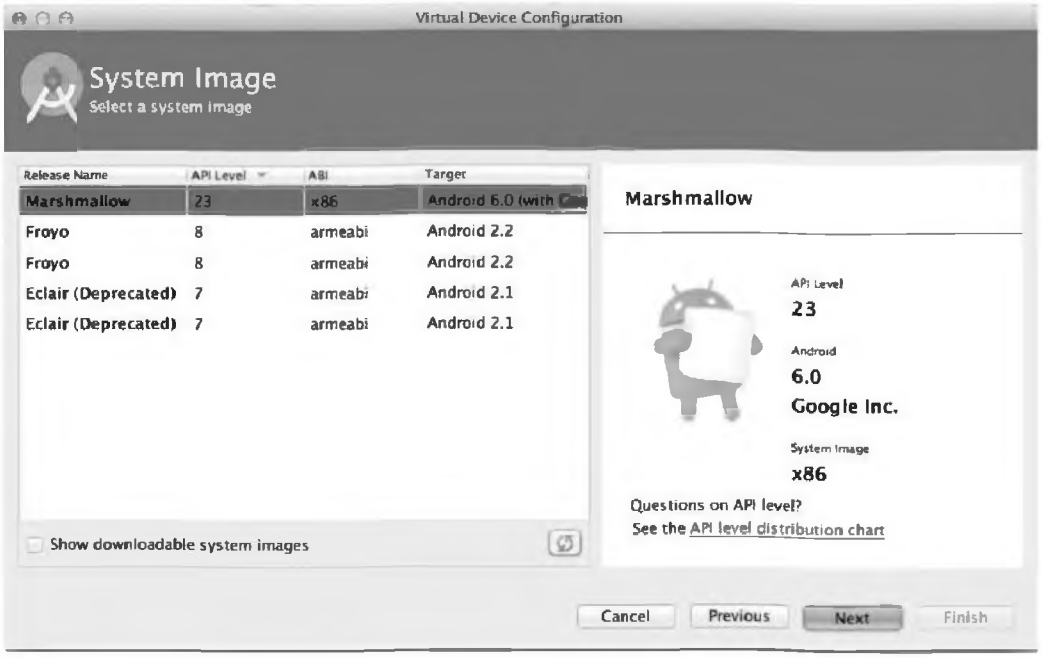


Figure A-72

Copyright © 2016 Eyrolles.

4. Nommez votre émulateur (ici, « The Nexus S API 23 ») et choisissez les paramètres par défaut pour l'orientation… Terminez en cliquant sur Finish.

Figure A-73

Remarque Le nom de l'émulateur ne doit pas contenir de caractères spéciaux (-, +, /…), ni d'espace.

5. La fenêtre Android Virtual Device Manager apparaît à nouveau avec l'item correspondant à votre émulateur (ici, « The Nexus S API 23 ») sélectionné. Fermez le panneau, pour accéder à l'IDE Android Studio.

6. Lors de la première compilation, le panneau Device Chooser apparaît. Sélectionnez la puce Launch emulator et le dispositif The Nexus S API 23 dans la liste proposée. Pour finir, cliquez sur OK.

Figure A-74

Copyright © 2016 Eyrolles.

© Éditions Eyrolles

7. La fenêtre d'émulation du smartphone apparaît, vous pouvez vérifier le bon fonctionnement de votre application.

Figure A-75

Votre environnement de programmation d'applications Android est prêt. Pour créer et développer vos propres applications, reportez-vous au chapitre 13, « Développer une application Android ».

Copyright © 2016 Eyrolles.

Copyright © 2016 Eyrolles.

Index

Symboles

.class 25
.java 24
= 50

A

accès
  données 211
  en consultation 238, 284, 313
  en modification 208, 238
  méthode 212
accesseur 238
accumulation 112, 115, 122, 133
action 358
actionPerformed 408
adresse 10, 15, 192, 203, 233
affectation 47, 51, 63
afficher 16
algorithme 5, 8, 31
  paramétré 144, 161
analyse descendante 6
Android
  AndroidManifest 502
  background 493
  Bundle 487
  CheckBox 499, 503
  drawable 485
  fill_parent 482
  finish() 505
  id 493
  ImageView 499
  Intent 501
  isChecked() 504
  layout 483, 487
  Log.i() 505
  openFileInput() 506
  orientation 499
  read() 506
  setChecked() 507
  setContentView() 487
  setOnClickListener() 496
  tActivity 489
  text 494
  tfindViewById() 496
  Toast 497
  values 485
  view 490
  wrap_content 483
  write() 505
Android Studio 476, 535
  installation 576
Android:paddingBottom 484
Android:paddingLeft 484
Android:paddingRight 484
Android:paddingTop 484
application 209
  exemple 212, 230, 236, 239
  multifichier 213
archivage 324
ArrayList 437
  syntaxe 306
  Voir liste 306
ASCII 42
attribut 216
AWT package 308

B

bibliothèque
  Voir package 308
binaire
  Voir code 22
bloc 21
  if-else 98, 104
  instruction 88, 110, 143, 167, 171, 185

Copyright © 2016 Eyrolles.

Bloc-notes 329
boolean 40
bouton
  Voir Classe Button 359
break 101, 104
  Voir switch 98
ButtonGroup 445
  add() 446
byte 43, 100

C

Canvas *Voir* classe
  exemple 352
case
  mémoire 10, 15
  Voir switch 98
cast 58, 61, 121, 129, 309, 332
  exemple 58
catch 336
  Voir Exception 334
char 40, 49, 63
classe 19
  abstraite 407
  ArrayList 307
    add() 307, 308
    clear() 307
    get() 307, 309
    indexOf() 307, 311
    lastIndexOf() 307
    remove() 307
    size() 307
  BufferedReader 325
    readLine() 328
  BufferedWriter 325
    close() 328
    newLine() 327
    write() 327
  Button 359
  Canvas 352
    exemple 357
    paint() 353
    setbackground() 353
    setCursor() 353
    setForeground() 353
Checkbox 381
Collection 315
Color
  brighter() 356
  darker() 356
  différentes couleurs 381
  setColor() 357
  white 353
définir 205
dérivée 247
Frame 351
  add() 354
  addWindowListener() 367
  setBackground() 351
  setSize() 351
  setTitle() 351
  setVisible() 351
Graphics 353
  drawString() 386
  fillOval() 381
  fillPolygon() 354
  fillrect() 386
HashMap
  get() 312, 314
  put() 312, 314
  remove() 312, 315
  size() 312
  values() 315
Integer
  parseInt() 124, 126, 281
  toString 130
Iterator
  hasnext() 315
  next() 315
JFrame
  getContentPane() 371, 377
JPanel
  paintComponent() 372, 439
Locale 79
Math
  abs() 145
  ceil() 145

Copyright © 2016 Eyrolles.

cos() 145
exemple 146
exp() 145
floor() 145
log() 145
max() 145
min() 145
PI 17
pow() 82, 145, 149
random() 135, 145, 148, 296, 355
sin() 145
sqrt() 82, 105, 145, 148, 150
tan() 145
ObjectInputStream 331
close() 332
readObject() 332
ObjectOutputStream 331
close() 332
writeObject() 331
Panel
add() 360
addActionListener() 364
exemple 360
repaint() 366
Runnable 406
Scanner 16, 77, 120
next() 81
nextInt() 81
String 122, 191
charAt() 194, 196, 282, 313
compareTo() 198, 200
concat() 201, 202
endsWith() 194, 196, 197
equals() 198
equalsIgnoreCase() 198, 200
exemple 195, 198, 201
indexOf() 194, 196, 197
lastIndexOf() 194
length() 201, 202, 327
regionMatches() 198, 199
replace() 201
startsWith() 194
substring() 194, 196, 218
toLowerCase() 201, 202
toUpperCase() 201, 202, 313
valueOf() 327
StringTokenizer 343
super 247, 439
System
.in.read() 120, 121
.out.print() 16, 71, 81
.out.println() 73, 81
err 70
exit() 84, 368, 428
TextField 387
getText() 387
UIManager
setLookAndFeel() 376
WindowAdapter
windowClosing() 368
CLASSPATH 213
Linux 550
Windows 545
clé
Voir dictionnaire 311
cmd.exe 75
code
binaire 13, 22
page 42, 75
pseudo- 22, 24, 213, 214, 226, 231
source 22, 134
Unicode 128, 129
code-point 41
collection
stream 322
commande 25
java 23, 25
javac 23, 24
commentaire 18
compilateur 22, 31
compilation
multifichier 213
comportement 207, 216, 235, 283
comptage 112, 133
concaténation 72, 122
condition 90
constante 241, 450

Copyright © 2016 Eyrolles.

constructeur 285, 289
exemple 244, 245
par défaut 243, 249
surcharge 250
conteneur
Voir classe Panel 360
contrôle des données
exemple 238
couleur 450
Cp1252 42, 75

D

déclaration
objet 209
variables 15, 16
default 102, 104, 254, 256
Voir switch 98
Design 396, 402
dictionnaire 311
clé 312, 321
exemple 319
créer un dictionnaire 313
créer une clé 313
rechercher un élément 314
supprimer un élément 314
dispose() 433
do...while 111, 132, 310
choisir 131
exemple 117
syntaxe 112
DOS 70, 75
Double
parseDouble() 409
toString() 409
double 16, 43, 63, 148
drapeau 459

E

échanger 289
des objets 229
des valeurs 51
exemple 51
EDI (Environnement de développement intégré) 389
encapsulation 235
entrée-sortie 12, 69, 81
erreur
cannot find symbol class scanner 20
class not found 213, 308
else without if 95
expected 158
FileNotFoundException 327, 334
Identifier expected 160
Incompatible type for =. Explicit cast... 58
Incompatible type for method 149
java.lang.ArrayIndexOutOfBoundsException 276
Java.lang.NumberFormatException 125, 126
java.util.InputMismatchException 79
no constructor matching 249
NotSerializableException 330
undefined variable 170
UnsupportedEncodingException 75
variable in class not accessible 250
variable is already defined 298
variable is already defined in this method 154
variable may not have been initialized 47
variable not accessible from class 237
étiquette 101, 104
Voir switch 98
événement 358
Action 423
bas niveau 362
gestionnaire 408, 410, 423
haut niveau 363
mouseEntered 426, 427
mousePressed 440
mouseReleased 440
survol 425
Voir Interface 362
EventListener 362
exception
ClassNotFoundException 332, 336
exemple de capture 335
IOException 75, 126, 326
plaf 376

Copyright © 2016 Eyrolles.

exemple
affectation 48
calcul de statistiques 62
cast 58
compter des cercles 227
compteur de monnaie 117
constructeur par défaut 244
contrôle du rayon 242
de cercle
contrôlé 238
et fonction 156
objet 207, 212
protégé 235
simple 19
déclaration 46
déclaration de tableaux 273
fonction max() 159
fonctions mathématiques 146
gestion d'exception 335, 336
héritage 247
la classe
Arbre 355
Cursus 285, 307
DesBoutons 360, 364
DesBoutonsSwing 373
Dessin 352, 357
DessinSwing 372
Etudiant 283
Fenetre 350, 359
FenetreSwing 369
GestionAction 364, 374
GestionClasse 289
GestionCursus 309, 319, 332
GestionFenetre 367
GestionFichier 328
String 195, 198, 201
Triangle 353, 356
ligne de commande 280
lire un entier 125
nombre de jours par mois 100
passage par valeur 179
quel code Unicode ? 130
résultat d'une fonction 181
trouver le plus grand nombre 94
un sapin en mode caractère 296
variable
de classe 173
locale 171
visibilité 169
exit() 102, 296, 367
extends 246, 248, 255, 352, 360
exemple 247

F

fenêtre 350
fichier
d'objets 329, 433
exemple 328, 332
ouvrir 325
texte 325, 437, 453
File
getSelectedFile() 423
toString() 423
final 241
Float
parseFloat() 432
toString() 432
float 43
flux 324
de fichier 324
entrant/sortant 324, 325
fonction 144
appel 150
corps 152
définition 151
en-tête 153
exemple 156, 159, 160
main() 19, 156, 209, 406
en-tête 278
nom 147, 152
paramètre 148, 153, 159, 185
résultat 148, 158, 185
sans paramètre 160
sans résultat 158
type 155, 161

Copyright © 2016 Eyrolles.

for 127, 133, 275
choisir 132
exemple 130
imbrication 292, 297
syntaxe 127
frame
exemple 350

G

gestionnaire d'événements 408, 410, 423, 440
get 240
getBytes() 75
getProperty() 74
getSelectedFile() 423
getSelectedItem() 424
getText() 409, 421
Graphics
fillOval() 441

H

Hamcrest 559
HashMap
exemple 319
syntaxe 312
Voir dictionnaire 311
héritage 246
hexadécimale 43, 129

I

i-- 128
i++ 128
if-else 89, 103
bloc 98
choisir 102
erreur 95
exemple 94
imbrication 96
syntaxe 89
ImageIcon 430
implements 253, 365
ActionListener 374
Serializable 330
import 20, 77, 351
incrémentation 50, 115, 127, 128, 133, 226, 229
indice
Voir tableau 275
initComponents() 408
initialisation 49, 121
Inspecteur 399, 402
installation
Android Studio 576
Java SE 536, 545
NetBeans 550, 555, 561
instance 211
instructions 9, 10
int 43, 63, 114, 120
interface 252, 365, 407
ActionListener 363
actionPerformed() 363, 364, 366
exemple 364
default 254, 256
graphique 358, 389
ItemListener 381
itemStateChanged() 382
MouseListener 362
mouseClicked() 362
mouseEntered() 362
mouseExited() 362
mousePressed() 362
mouseReleased() 362
MouseMotionListener 362
mouseDragged() 362
mouseMoved() 362
WindowListener 363, 367
exemple 367
windowActived() 363
windowClosed() 363
windowClosing() 363, 367
windowDeactived() 363
windowDeiconified() 363
windowIconified() 363
windowOpened() 363
interpréteur 22, 31, 214
invokeLater() 406

Copyright © 2016 Eyrolles.

J

Java
  paramètre 279
java 214
Java SE
  installation 536, 545
java.util 20, 77
javac 213
JButton 404
  exemple 374
  setToolTipText() 447
JComboBox 416, 424, 459
  getSelectedItem() 424
  model 417
  PopupWillBecomeVisible 459
  PopupWillBecomInvisible 460
  selectedIndex 417
JDK 23, 349, 535
jeu de caractères 74
JFileChooser 421
  APPROVE_OPTION 423
  showOpenDialog() 422
  showSaveDialog() 462
JFrame 394, 418
  defaultCloseOperation 414, 419
  DISPOSE 419
  dispose() 433
  exemple 370
  EXIT_ON_CLOSE 414
  title 398, 414, 419
JLabel 401
  background 415
  font 415
  setIcon 430
  setText() 430
  text 414
JMenuBar 435, 451
JMenuItem 451
JPanel 418, 435, 438
  exemple 372, 373
  paintComponent() 439
  repaint() 439, 443
JTexField
  Right 403
JTextField 403
  getText() 409, 421
  horizontalAlignment 403
  setText() 409
JToggleButton 445
  setIcon() 446
  setSelected() 446
JToolBar 435, 445
JUnit 559
JVM 22

L

layout 484
length 294
  Voir tableau 274
length()
  Voir String 202
ligne de commande 278
Linux 23, 76
  CLASSPATH 550
  PATH 550
liste
  exemple 307, 309
Locale.FRENCH 79
Locale.US 79
long 43

M

Mac OS 23, 121, 279
  TextEdit 329
main() 406
mémoire centrale 9, 31
méthode 144, 194
  d'implémentation 243
  get 240
  invisible 242, 284, 288
    exemple 242
  métier 243
  next() 79
  nextLine() 79

Copyright © 2016 Eyrolles.

set 240
useLocale() 79
modulo 53
Mouse
mouseEntered 426
mouseEntered 426, 427, 440
mouseReleased 440

N

NetBeans 390, 535
actionPerformed 408
Aligner 404
Anchor 404
Ancre 404
code généré 405
Design 396, 402
Événements 408
Event 408
initComponents() 408
Inspecteur 399, 402
installation 550, 555, 561
main() 406
Même Taille 404
Palette 398
plug-in 559
projet 391, 392, 570
Propriétés 398, 403
Source 396
new 210, 248, 273, 285, 291
null 210, 314, 329

O

objet 210
définition 193
octet 39
opérateur 63
!, &&, || 92
+, -, *, /, % 52
logique 92, 103
priorité 53
relationnel 90, 103
override 487

P

package 20, 77, 308, 394
java.awt 351
java.awt.event 364
java.io 324
java.util 308, 315
javax.swing 370
paintComponent() 439
Palette 398
panel, exemple 364
panneau
Design 396, 402
Inspecteur 399, 402
Palette 398
Propriétés 398, 403
Source 396
paramètre 142
formel 154, 179, 181, 230, 366
objet 232
passage
par référence 229, 234
par valeur 179, 181
réel 182
effectif 154
parseDouble() 409
parseFloat() 432
PATH
Linux 550
Windows 538
PC 121
plaf 372, 376
plug-in
Hamcrest 559
JUnit 559
polymorphisme 250
principe de fonctionnement
do...while 112
fonction 147
for 127
gestion d'un événement 368
if-else 89
imbriqués 96

Copyright © 2016 Eyrolles.

ouvrir un fichier
en écriture 327
en lecture 326
switch 98
tableau 274
try/catch 335
variable de classe 178
while 119
principe de notation
objet 205, 241, 420
priorité 63
programmation dynamique 305
propriété 216
Propriétés 398, 403
protected 250
protection des données
exemple 235
private 235, 242
protected 235
public 235
pseudo-code
Voir code 24
public 278

R

raccourci clavier 452
référence 193, 203
relation est un 246, 258
RelativeLayout 484
repaint() 439, 443
réservation d'un espace mémoire
Voir new 210
return 154, 158, 159, 161, 181, 184
Runnable 406

S

sérialisation
Voir fichier d'objets 329
set 240
setIcon() 446
setSelected() 446
setText() 409
setToolTipText() 447
short 43, 100
showOpenDialog() 422
showSaveDialog() 462
Source 396
static 174, 209, 226, 278, 330, 450
stream 322
Voir flux 324
String 279
getBytes() 75
replace() 504
StringTokenizer 507
structure d'un programme 20, 168
super 249, 251, 267, 373, 439
surcharge 245, 487
de constructeur 250
de méthodes 243, 313
swing
Voir package 308
switch 98, 100, 104, 115
choisir 102
exemple 100
String 99, 200
syntaxe 98
syntaxe
définition 14
System
exit() 102, 296, 367, 428
getProperty() 74
in.read() 76
out.println() 69, 130

T

tableau 272
2 dimensions 291
d'objets 285, 299
déclaration 272, 291, 299
exemple 276, 285
indice 275, 292, 294
initialisation 277
length 274, 276, 291
taille 274, 281

Copyright © 2016 Eyrolles.

taille
  tableau 274
TextView 484
this 246, 365, 433
throws 334, 335
  Voir Exception 75, 125, 326
title 414
toString() 409, 423
tri par extraction 286
try
  Voir Exception 334
type 204
  choisir 44
  conversion 57
  définition 39
  générique 437
  objet 193, 209
  simple 193
  structuré 40, 206, 209
type de données 12

U

Unicode 41
  code-point 41
  Voir code 73
unité centrale 9, 31
Unix 22, 24, 25, 70, 121, 279
  vi 329

V

variable
  d'instance 211, 227, 430
  de classe 173, 175, 176, 177, 185, 227
  déclaration 45, 48, 169, 272
  définition 38, 63
  invisible 170
  locale 170, 172, 175, 185
  static 174, 226, 450
  tableau d'évolution 123, 171, 175, 234, 294
  véritable nom 177
visibilité 185
void 159, 161, 278

W

while 119, 121, 133
  choisir 131
  exemple 125
  syntaxe 119
Windows 23, 25, 279
  Bloc-notes 329
  CLASSPATH 545
  PATH 538

X

XML 484

© Éditions Eyrolles